NO. 32 $2.00 JANUARY 1981

MICRO™

THE 6502 JOURNAL

AIM 65 File Operations

Fun With OSI

PET Symbolic Disassembler

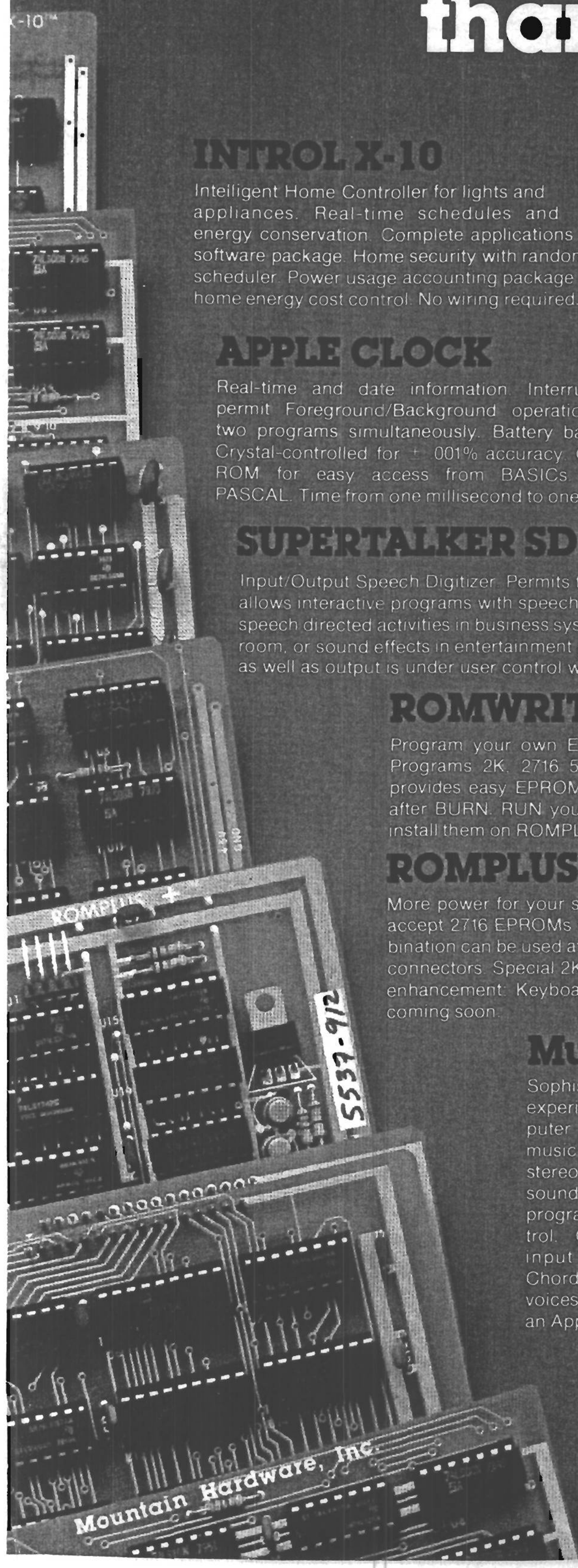
Atari Real Time

SYM Bridge Trainer

A Better Apple Search/Change

Mountain Computer makes more peripherals for the Apple Computer than Anybody.

and . . . a place to put them



INTROL X-10

Intelligent Home Controller for lights and appliances. Real-time schedules and energy conservation. Complete applications software package. Home security with random scheduler. Power usage accounting package for home energy cost control. No wiring required.

APPLE CLOCK

Real-time and date information. Interrupts permit Foreground/Background operation of two programs simultaneously. Battery back-up. Crystal-controlled for ± .001% accuracy. Onboard ROM for easy access from BASICs. Supports PASCAL. Time from one millisecond to one year.

SUPERTALKER SD200

Input/Output Speech Digitizer. Permits talking programs. I/O capability allows interactive programs with speech-prompted inputs. Use output for speech directed activities in business systems, announcements in a control-room, or sound effects in entertainment programs. Easy to use because input as well as output is under user control with special software operating system.

ROMWRITER

Program your own EPROMs. Create your own firmware. Programs 2K 2716 5V EPROMs. Disk software package provides easy EPROM programming. EPROMs are verified after BURN. RUN your programs from on-board socket or install them on ROMPLUS+.

ROMPLUS+

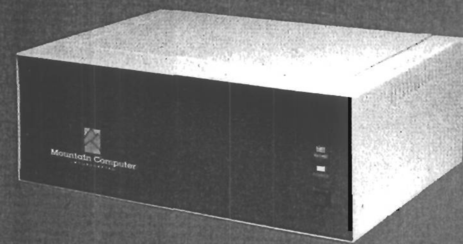
More power for your system through firmware. Six sockets accept 2716 EPROMs or ROM equivalents. Six or any combination can be used at once. Scratch-pad RAM and two TTL connectors. Special 2K ROMs available for powerful system enhancement: Keyboard Filter ROM—COPYROM—Others coming soon.

MusicSystem

Sophistication previously available only on experimental mini and mainframe computer synthesizers. Digital instrumental music synthesizer system. 16 voices in stereo. Instrument definitions simulate the sound of real instruments—and more. Fully programmable waveforms. Envelope Control. Composition system—sheet music input using standard music notation. Chords and multi-part scoring up to 16 voices. A true instrument that anyone with an Apple can play.

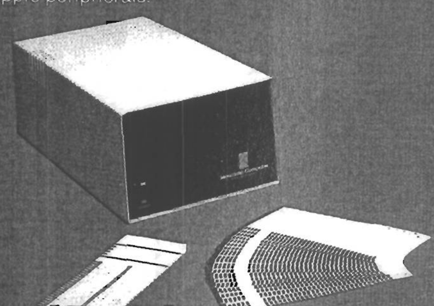
A/D+D/A

16 channels analog to digital input. 16 channels digital to analog output. Eight bit resolution. Super-fast 9μ sec. conversion time. Monitor and output to the real world. All on one card.



EXPANSION CHASSIS

By popular demand! Eight more slots for your Apple. Attractive sturdy enclosure. Its own heavy duty power supply. Easy to use. Address cards in Expansion Chassis the same way as in your Apple. Only one additional command to specify in Apple or in Expansion Chassis. Compatible with all Apple peripherals.



CardReader

At last! An intelligent, high-quality device for data entry from user-marked cards. Implement BASIC programming, examination scoring, inventory maintenance and other applications requiring off-line data preparation for batch entry later. Connects to any computer having RS-232 interface. Software and cards are available for jobs in business, science and education.

MOUNTAIN COMPUTER* has the most comprehensive line of Apple peripherals available. Anywere. From anybody. We know the Apple inside and out and are committed to providing the most innovative and unique products to expand and enhance its capabilities and use. After all, we were the first company to make an Apple peripheral—except Apple Computer.

Available at Apple Dealers worldwide.

Mountain Computer
INCORPORATED
300 Harvey West Blvd., Santa Cruz, CA 95060
(408) 429-8600 TWX 910 598-4504

*Based on Manufacturer's published catalogs—Apr. 1980

Apple is a trademark of Apple Computer Inc.

NO. 32 JANUARY 1981

MICRO™

THE 6502 JOURNAL

CONTENTS

7 KEYBOARD ENCODING
How to connect a keyboard to your computer
By George Young

17 A BETTER APPLE SEARCH/CHANGE
Remove the length restriction of the CHANGE function
By J.D. Childress

21 VECTORS AND THE CHALLENGER 1P
Find out exactly what a vector is
By Mike Bassman

23 PET SYMBOLIC DISASSEMBLER
Disassembled programs become more understandable
By Werner Kolbe

29 AIM 65 FILE OPERATIONS
The third part of this series, using BASIC
By Christopher J. Flynn

35 ATARI REAL TIME
Drawing with graphics and several colors
By Charley and Mary Kozarski

37 FULL DISASSEMBLY LISTING ON SMALL SYSTEMS
Produce "camera-ready" listings without large memory
By Ralph Tenny

41 SYM BRIDGE TRAINER
Simple program, easily adapted for any 6502
By Len Green

53 MAKE A CLEAR PLASTIC COVER FOR YOUR APPLE
Protect your Apple, and watch while you compute
By E.J. Neiburger, D.D.S.

57 SEARCHING STRING ARRAYS
Searching a large string array becomes easier
By Gary B. Little

65 INTERFACING THE 6522 VERSATILE INTERFACE ADAPTER
This chip will allow you to add more capabilities to your system
By Marvin L. DeJong

75 FUN WITH OSI
Checker game using the polled keyboard technique
By Les Cain

DEPARTMENTS

5 Editorial — MICRO Goes to School — Robert M. Tripp
6 Letterbox
36 New Publications
51 PET Vet — Loren Wright
61 MICRO Club Circuit
79 Semi-Annual Index
87 The MICRO Software Catalog: XXVIII
90 6502 Bibliography: Part XXVIII — William R. Dial
95 Advertisers' Index

STAFF

Editor/Publisher
ROBERT M. TRIPP

Associate Publisher
RICHARD RETTIG

Associate Editor
MARY ANN CURTIS

Art Director
GARY W. FISH

Typesetting
EMMALYN H. BENTLEY

Director Marketing
JAMES ANDERSON

Advertising Manager
L. CATHERINE BLAND

Circulation Manager
CAROL A. STARK

MICRO Specialists
APPLE: FORD CAVALLARI
PET: LOREN WRIGHT
OSI: PAUL GEFFEN

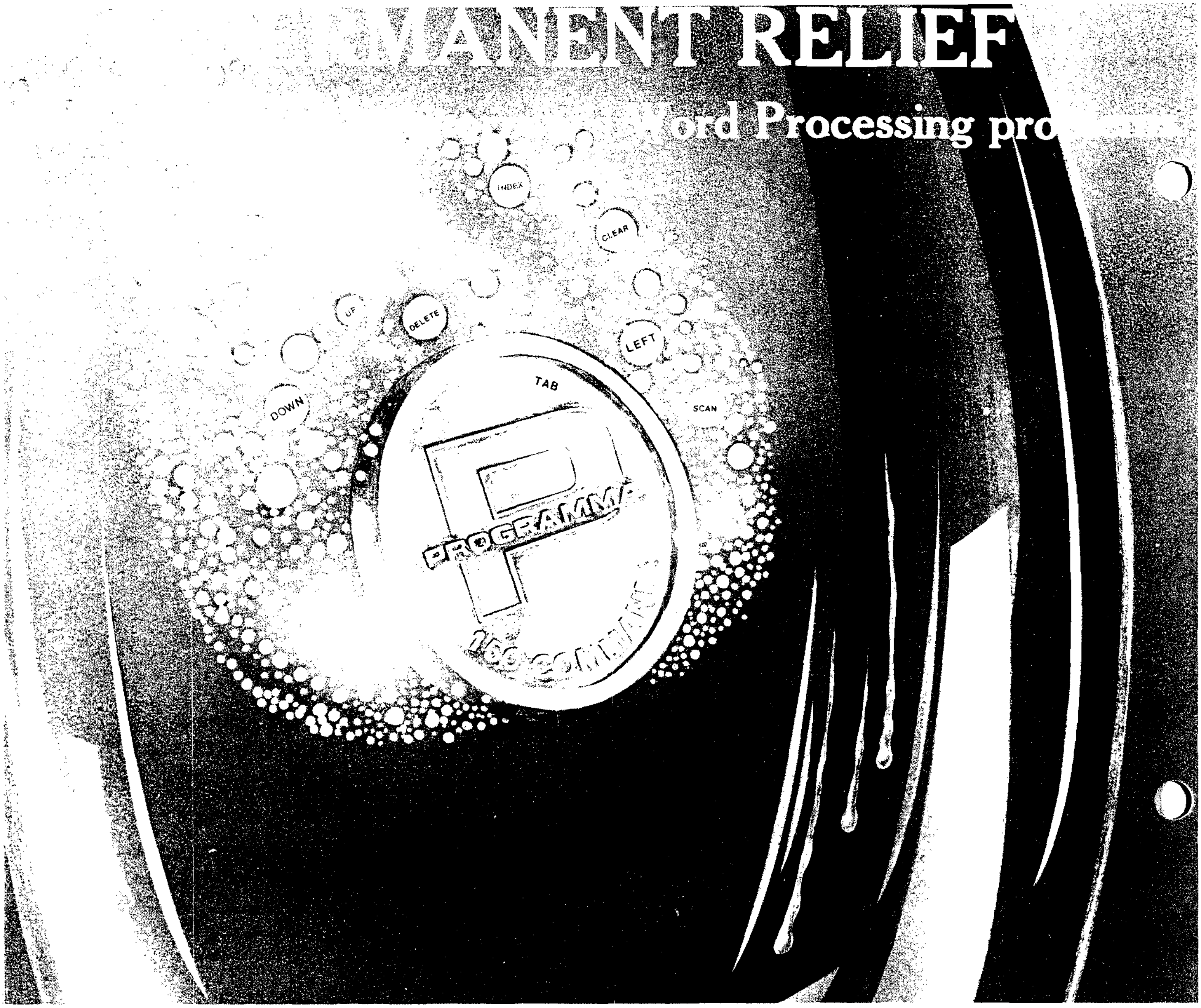
Comptroller
DONNA M. TRIPP

Bookkeeper
KAY COLLINS

MICRO is published monthly by:
MICRO INK, Inc., Chelmsford, MA 01824
Second Class postage paid at:
Chelmsford, MA 01824
Publication Number: COTR 395770
ISSN: 0271-9002

Subscription rates: U.S.	$15.00 per year
Foreign surface mail	$18.00 per year
Central America air	$27.00 per year
So. Amer/Europe air	$33.00 per year
Other air mail	$39.00 per year

For back issues, subscriptions, change of address or other information, write to:
MICRO
P.O. Box 6502
Chelmsford, MA 01824



Apple PIE + Formatter

Apple PIE (Programma International Editor) and FORMAT (text formatter) offer full strength solutions to today's word processing problems. These versatile, powerful programs provide document preparation and word processing capabilities previously found only on much larger computer systems.

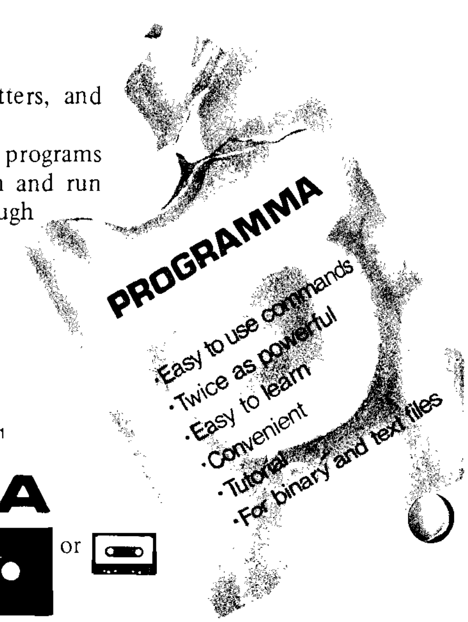
PIE is a general purpose, full screen editor that uses control keys and function buttons to provide a full range of editing capabilities such as search and replace, delete, copy, insert, move. Changes may be made directly anywhere on the screen and are shown as they are performed.

FORMAT uses simple instructions embedded in the input text to describe the desired appearance of the final document. It handles centering, underlining, indenting, page numbering, margins, headers, footers, even form letters, and includes a proofing capability.

These high-quality, cost-effective programs come with comprehensive documentation and run on a 32K Apple II. They are available through your local computer store or direct from Programma International, Inc. at the introductory price of $79.95*.

VIDEX VERSION T.M.
DOUBLE VISION T.M.
SUPR TERM VERSION T.M.
STANDARD VERSION

*December 1, $129.95.



PROGRAMMA

3400 Wilshire Boulevard
Los Angeles, California 90010
(213) 384-0579

or

Simple enough for the beginner. Versatile enough for the professional.

MICRO Editorial

MICRO Goes To School

Almost every progressive school now uses computers to teach about computers, to teach other subjects, and for administrative and other non-teaching tasks. Since a school may already own a mini- or maxi-computer, or have access to one within the town or school district, the assumption is often made that it will be cheaper to buy additional devices to tie into an existing system than to purchase new systems. This assumption may be erroneous since many of the operations can be more effectively performed by a dedicated microcomputer than by a large time-shared central computer.

A local high school rents 10 teletype terminals at over $500 each, per year, to access a medium-sized system at the regional vocational high school. One or two of the terminals are used for administrative tasks, which may require a large disk storage and/or voluminous print-outs that could be best handled by the central system. The remaining terminals are for students learning computer programming, especially programming in BASIC. Under this remote terminal approach, students can only use the teletype with its obvious limitations of speed and minimal characters, can only access the central computer at certain times, do not get the "hands-on" experience of actually operating a microcomputer, do not get the graphics and instant video response of the micro, and probably do not have the range of programs available—games and others—that they would get on a micro. Meanwhile, the school system is paying as much to rent one terminal for two years as it would cost to buy a complete 6502-based video system. This does not even include the hidden expenses of the telephone, support for the central system, special main-frame programming, and so forth.

This is absurd, but understandable. The large computer manufacturers have spent time, money and effort to convince the educational community of the value of a central time-sharing system. No microcomputer company has had the resources or time to tackle this monumental task alone. Advances to date have been mostly due to individual computer stores, "enlightend" teachers, and the occasional outside individual, getting the school systems to think along micro lines.

What better way to learn about a computer than to have the entire system at your disposal—no mysterious big black boxes in special air conditioned rooms located off-limits in another part of the building, or even across town. Probably more programs have been created for learning about computers using popular 6502-based microcomputers than were ever devised for any mainframe.

Beyond the teaching aspect of the micro in the school, there are numerous possibilities in school administration. Students could even write some of the programs. (Note: teachers had better carefully check the programs for accuracy. Some of the computer students are pretty sharp and might just "adjust" their grades!)

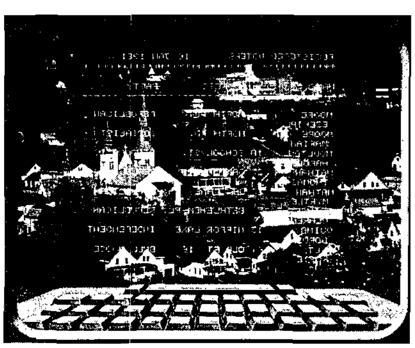
MICRO would like to try, with the aid of its readers, to help expand microcomputer usage in schools. There are several simple steps which you can take to help. First, become aware of what your local school system is doing in the computer field. Is it keeping up with the microcomputer revolution? If so, please let us know—in detail—how your school system is doing it. Perhaps the people responsible for the computers would be willing to write up some of their experiences, decisions, and so forth, providing assistance to other schools which are out-of-date.

If your school system is not keeping up, then offer to help. Find out who in your school system has the responsibility for the computer field and invite him or her over to see your equipment in operation. Show off some of your system's features and emphasize its low cost, its capabilities in the teaching area, and the fact that there is a lot of support through books and magazines such as MICRO. Explain that there are lots of programs available, which are generally inexpensive, covering many areas (show them the MICRO Software Catalog). Offer to help the school system evaluate its needs and requirements. You would be surprised at how small a push it may take to get things going. If there is enough reader interest, we would like to actively pursue this topic. This could include columns on educational applications of the 6502-based systems, special software listings of educationally related programs, discussions of school system needs, and so forth.

Robert M. Tripp

Editor/Publisher

About the Cover



MICRO Goes To Town

Our cover photograph, by Loren Wright, is of Littleton, New Hampshire. As we look at towns like this, the question arises, "How can a town or city effectively utilize microcomputers?" While some data collection and processing requirements of a government probably require the use of medium to large computer systems, other functions could be more efficiently and economically handled by a dedicated microcomputer.

Our cover shows a micro being used as a processor for voter registration. This simple application could include printing labels for the mailing of voter information, updating lists as voters are added, dropped, or change their addresses, recording changes in party affiliation, and so forth. Confidential material would not have to go outside the office; updating could be kept more current, and specialized lists easily prepared as required for a special election or other need; since this task would take only a small portion of the normal business week, the micro would be available to other departments and for other uses.

Other small data bases typically kept by local governments covering property assessments, zoning regulations, building permits, dog registrations, licenses, etc. could often be done on a microcomputer. Is your town or city keeping pace with the growing microcomputer capabilities?

MICRO Letterbox

Dear Editor:

Ohio Scientific C1P has a fine BASIC, but I envy the TRS-80 crowd and their PRINT USING. Since some programs must calculate and print dollar amounts, this simple routine puts the result of the calculation off to the right side of the screen, adds any required zeros and lines up the decimal points.

The only thing it does not do is round off amounts, and that would require only adding a line, DO = DO + .005 before transferring control to the subroutine.

Note that line 100 should be early in the program so the computer does not run across it each time the subroutine is called.

Zero amounts are printed correctly, but any dollar amount with more than six significant figures will cause the interpreter to go to scientific notation, which may give gross errors. Since I never have to worry about large amounts of money, that problem does not bother me.

```
100 Z$ = "00"
200 REM PROGRAM
    CALCULATES DOLLAR
    VALUE--DO
210 REM SUBROUTINE
    PRINTS
220 GOSUB10000
230 REM REST OF
    PROGRAM
    ....................
10000 DO$ = STR$(DO):IFDO
      = INT(DO)THEN DO$
      = DO$ + "."
10010 FORP = ITOLEN(DO $):
      IFMID$(DO$,P,1)
      = "."THEN10030
10020 NEXTP
10030 O$ = DO$ + Z$:O$
      = "$" + LEFT$(O$,
      P + 2)
10040 PRINTTAB(20 - LEN
      (O$))O$
10050 RETURN
```

Robert V. Davis
1857 South Fourth
Salina, Kansas 67401

Dear Editor:

The following program will disable the "LIST" command on Ohio Scientific ROM BASIC computers. Programs can be loaded and run, but not listed. Normally, I do not like non-listable programs and accept the challenge to try to list them. Thus, I offer your readers the challenge of listing this program after it has been run. If you need a hint, read MICRO 25:15.

```
10 FORX = 546TO586
20 READQ:POKEX,Q
30 NEXTX
40 POKE11,62:POKE12,2
50 X = USR(1)
80 DATA230,195,208,2,230,
   196,165,195,141
81 DATA51,2,165,196,141,52,
   2,173,20,0,201
82 DATA153,208,2,169,128,
   76,197,0,169,76
83 DATA133,188,169,34,133,
   189,169,2,133
84 DATA190,96
OK
```

Tim Finkbeiner
Earl Morris
Midland, Michigan

(continued on page 76)

Keyboard Encoding

It may occur sooner, or it may occur later, but you *will* want to connect a keyboard to your computer.

George Young
Sierra High School
Tollhouse, CA 93667

I will begin with a fairly simple encoder for 16 separate switches. Figure 1 gives the circuit. The 16 switches may be discrete switches or they may be 16 switches in a surplus or recycled calculator keypad.

The 16 switches are connected to the inputs of a 74150 1:16 multiplexer (MUX). Two 74161's or 74163's are used. One 74161 is used as a counter and the second is used as a register. A clock formed from ½ of a 7404 hex inverter is used to drive the 74161 clock inputs.

The outputs of the 74161 counter drive the MUX and the inputs to the 74161 register. The binary code presented to the MUX address inputs causes the MUX to scan each one of its inputs looking for a low. Closing one of the 16 switches connected to the input of the 74150 will cause this input to go low. The output of the 74150 will go high when a low is detected on one of its inputs. This high is time coincident, or time dependent, on the counter binary code at the instant of switch closure. This is the basic property of multiplexing.

The high out of the MUX is inverted with a 7404 hex inverter stage to produce a low on a key closure. This low is fed to the PE (preset enable) of the two 74161's. This low will cause the register 74161 to place the counter code that corresponds to the switch closed on its four output lines. This code will remain unchanged at the register outputs until another key is depressed. Thus, the code, or data out of the encoder is latched by the 74161 register stage.

The output code of the encoder can be seen by connecting 4 LEDs to the 4 register output lines. Figure 2 shows one way to connect the LEDs for this purpose. A logic probe may also be used to probe each output line to verify the code generated by the encoder.

When keyboards are encoded, a strobe pulse is also usually needed for the computer. Keyswitches bounce when connected to TTL chips, so a key debounce circuit is also usually required. Figure 3 shows a NAND gate section gating the clock and driving a 74122 retriggerable one-shot. This combination will debounce the key contacts and generate an active high and an active low strobe pulse.

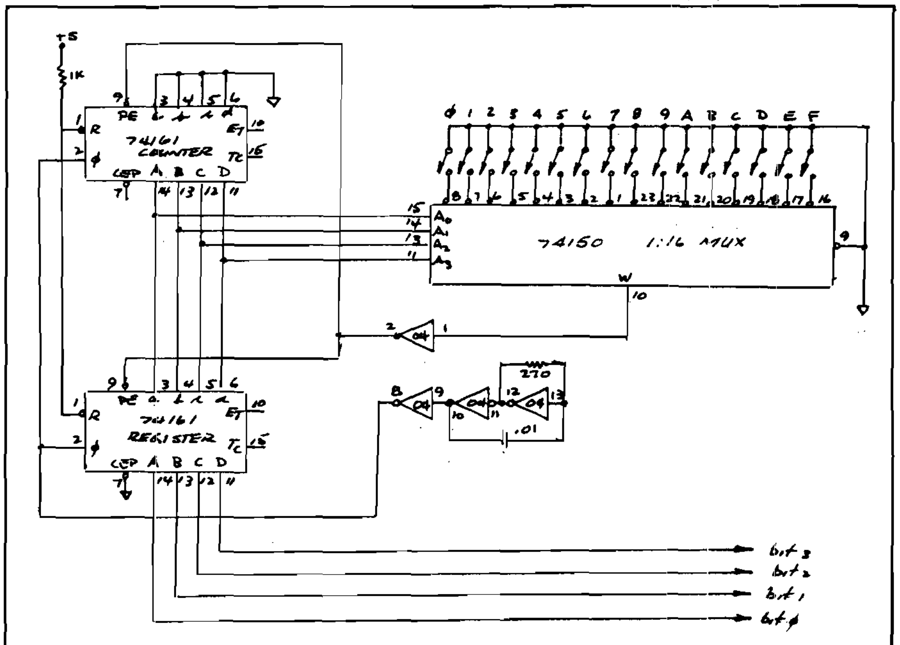


Figure 1. Encoder for 16 SPST switches.

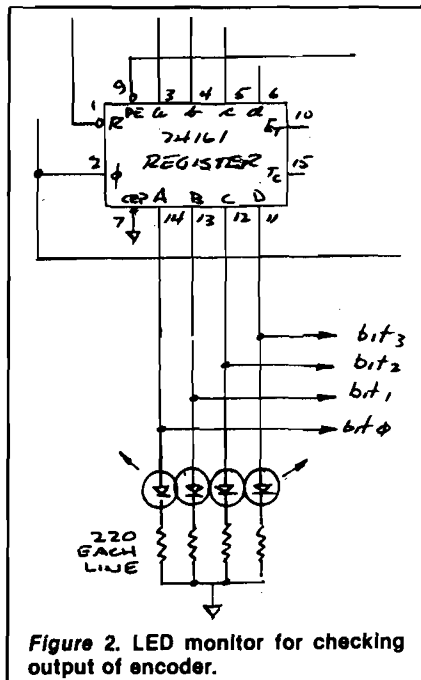


Figure 2. LED monitor for checking output of encoder.

To encode more than 16 keys, another 74150 would be needed with another 16 lines to the switch assembly. Additional 74161's used as counters and registers would be required to encode the added keyswitches. The major drawback to using this approach would be the size of the cable (minimum number of wires to the keyboard is 33; 32 for the keyswitches and one ground.) To circumvent this wire bundle, a different type of encoding is required.

Matrix Encoding

Only 8 wires need run from the switch assembly to the encoder circuitry to encode 16 switches, if the switches are arranged in a 4 × 4 matrix. The code is generated by a keyswitch matrix by using a 1:4 decoder to drive one set of 4 lines, which we will call the column lines, and feeding the other 4 lines to the inputs of a MUX. We will call these 4 lines the row lines.

The 74161 counter and register stages are again used in this circuit and they are again driven by the 7404 clock. Two of the counter outputs drive the two address lines of the decoder while the remaining two outputs drive the address inputs of the MUX. A 1:4 decoder (74139) or a 1:10 decoder can be used, and if the unused address lines are grounded, the decoder will function as a 1:4 decoder. A 1:8 MUX is used, operating as a 1:4 MUX again by grounding the unused address lines.

The counter code driving the address lines causes the decoder to put out sequential and mutually exclusive lows on its output pins. These lows feed the column lines of the switch matrix. The row lines feed the input pins of the MUX, and the remaining two 74161 counter outputs drive the address inputs of the MUX. Again, the MUX will look at each of its inputs to see if one of them is low. Closing a switch in the matrix causes the active low output of the MUX to go low, and the active high output to go high. This active high or low out of the MUX is again time coincident with the binary code, operating both decoder and MUX so the 74161 register outputs will again latch this code and you can examine it with LEDs on the 74161 register output lines, or with a logic probe.

Debouncing of the keyswitches and generation of strobes is again done with a NAND gate section and a 74122 retriggerable one-shot. All of this circuitry is shown in figure 4.

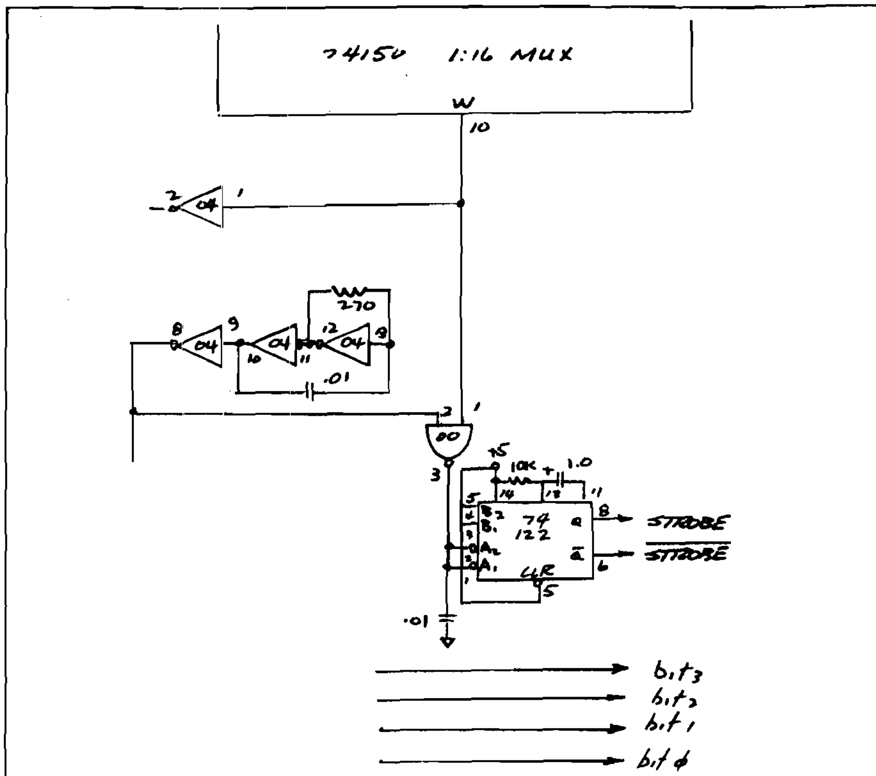


Figure 3. Key contact debounce/strobe generator.

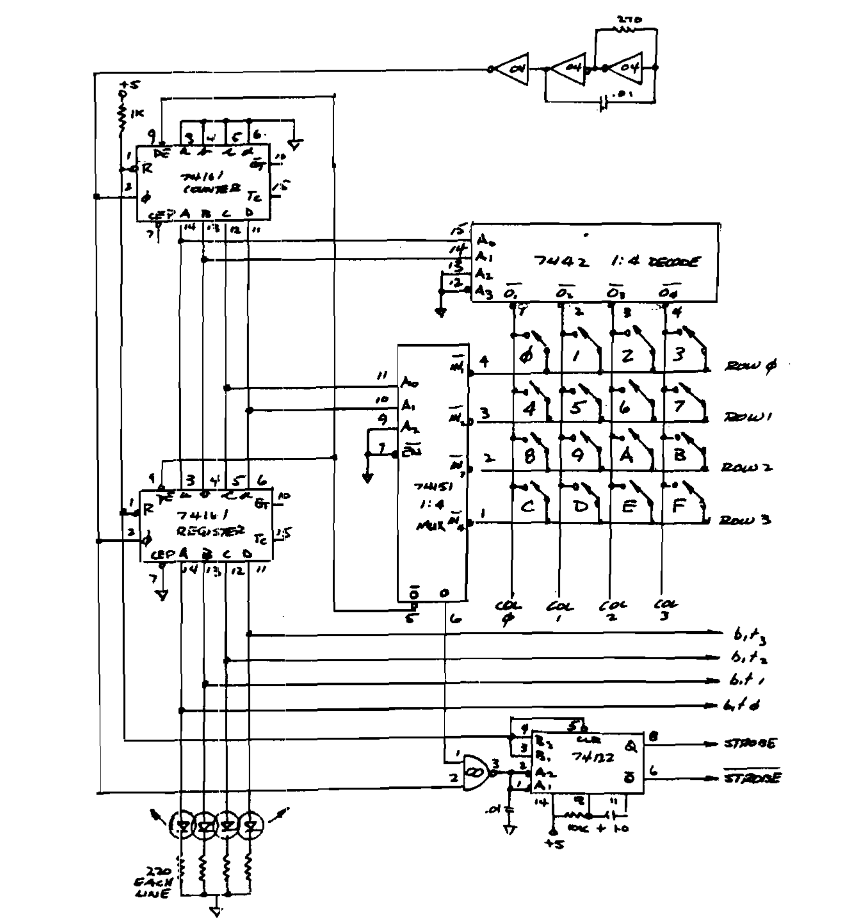


Figure 4. Basic matrix encoder for 16 switches.

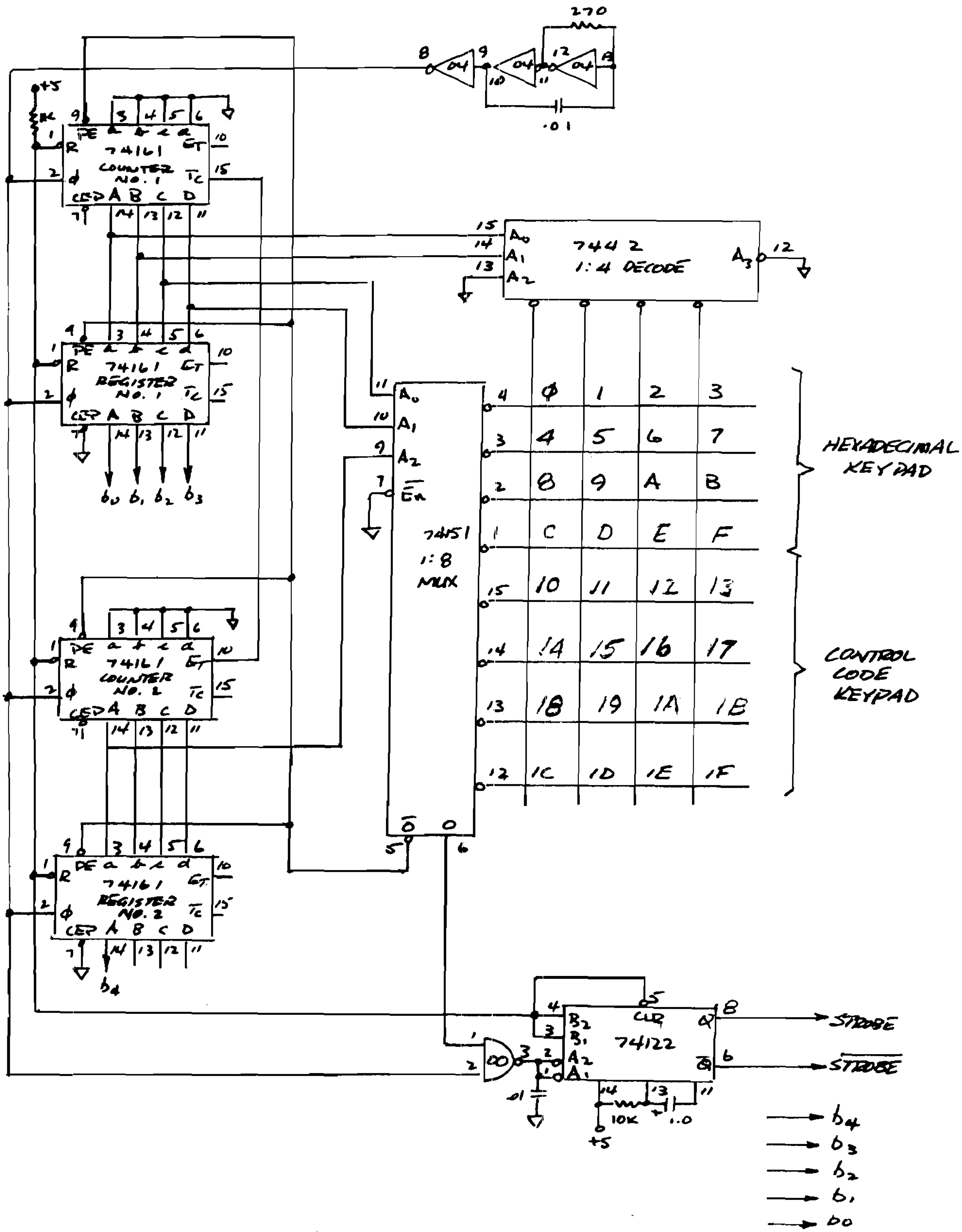


Figure 5. 32 key matrix encoder.

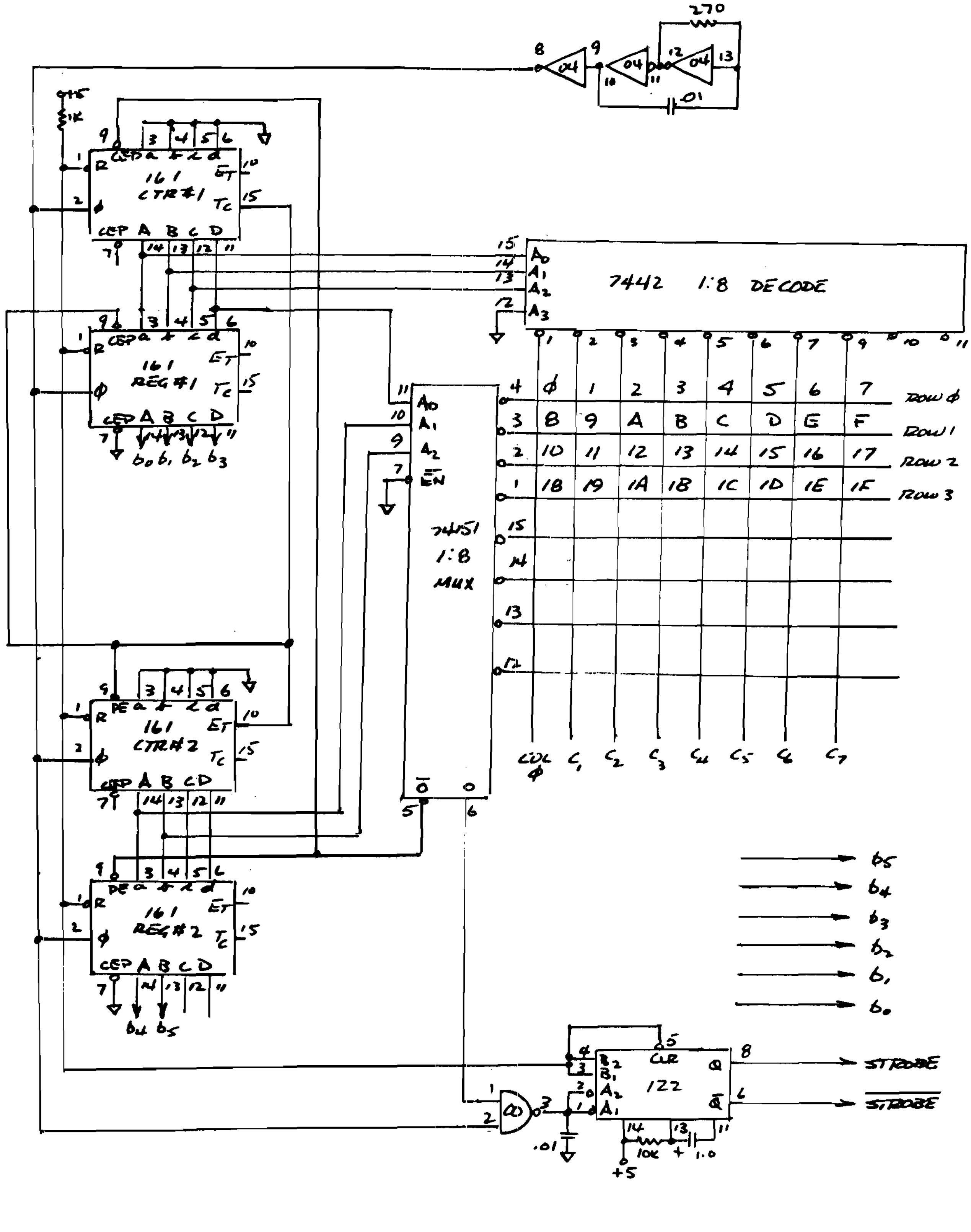


Figure 6. Encoding up to 64 keys on an 8 × 8 matrix.

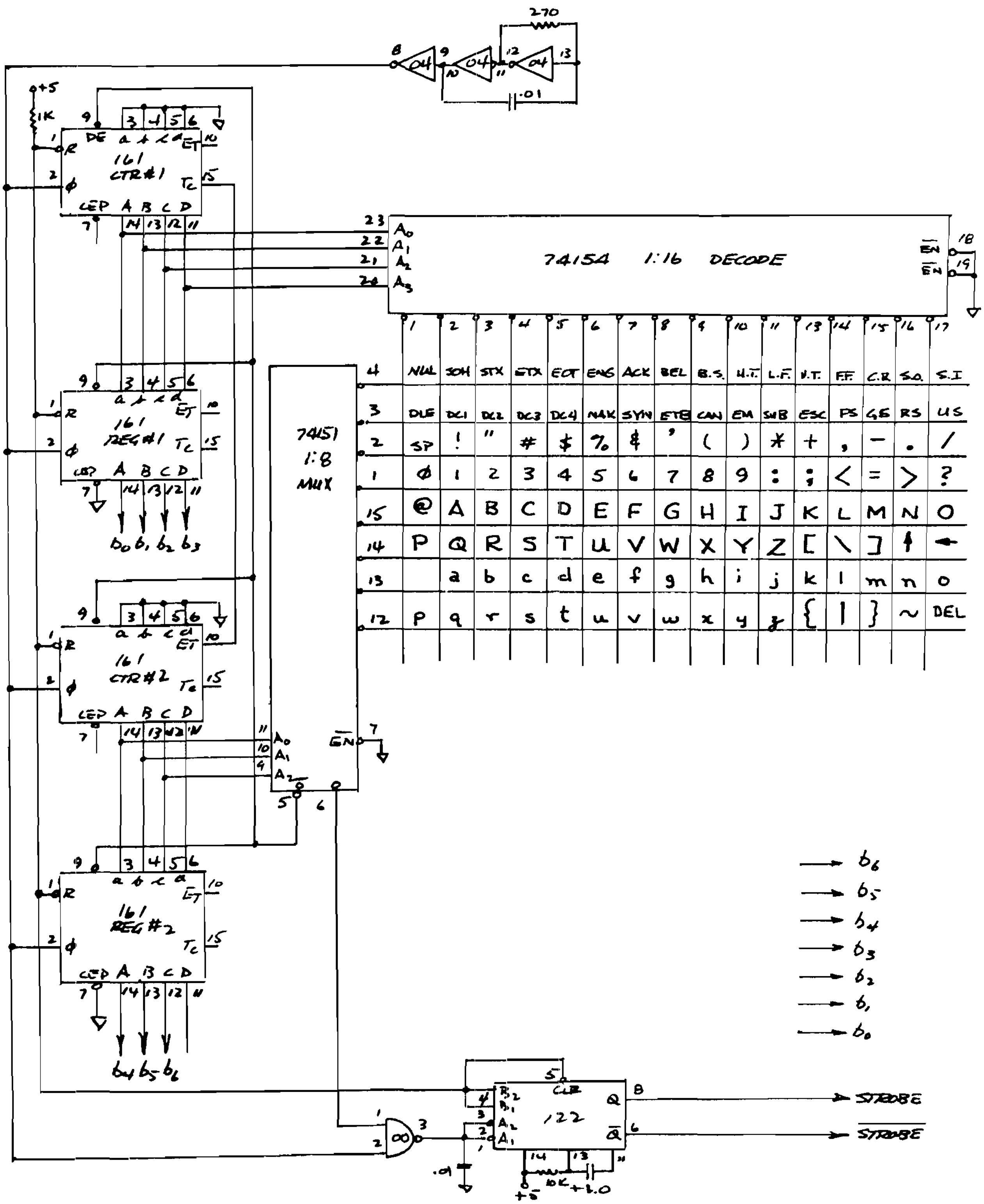


Figure 7. **128 key matrix encoder.**

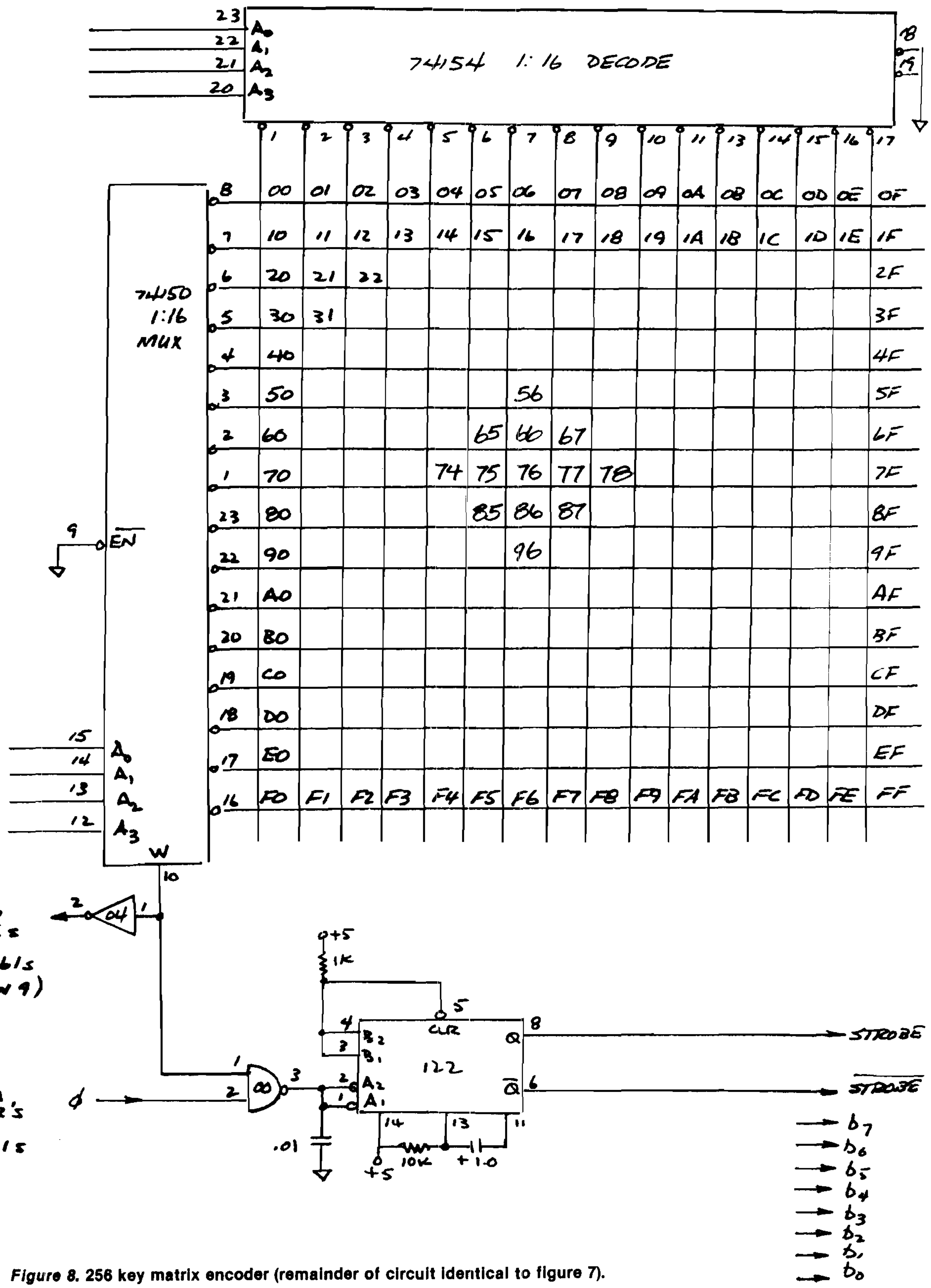


Figure 8. 256 key matrix encoder (remainder of circuit identical to figure 7).

Expansion

To encode 32 keys in a matrix requires only 4 additional lines from the keyswitches to the encoder circuitry. An additional 74161 counter stage, and an additional 74161 register stage must be added to the basic circuit of figure 4. I will first place the matrix in a 4 × 8 configuration. This circuit is shown in figure 5. The top keyswitch matrix will generate the hexadecimal code from 00 to 0F. The lower keyswitch matrix will generate hex codes from 10 to 1F. The keyswitches can also be arranged in an 8 × 4 matrix to encode the keyswitches. This circuit is shown in figure 6. Figure 6 also reveals that the encoding capability of the 4-74161's and 1:8 decoder and 1:8 MUX is only partially used. Sixty-four switches in an 8 × 8 matrix requiring 16 wires from switches to encoder circuitry can be encoded by the circuitry of figure 6.

The basic circuit is very flexible. Let us next encode a keyboard that has over 100 keys. We must change the decoder to a 74154 1:16 decoder, and place the keyswitches in a 16 × 8 matrix. In the configuration of figure 7, 128 switches can be encoded with the basic circuitry used earlier. The ASCII character set has been placed in position in the matrix to illustrate how the circuitry can be used to encode an ASCII keyboard. The builder will find, however, that he must make special provisions for the ASCII keyboard Shift and Control keys.

The circuit of figure 7 will encode up to 128 keys placed in a 16 × 4,5,6, 7 or 8 matrix and therefore can be used to encode almost any surplus keyboard. If the key in the matrix generates a different code than desired, it is a fairly simple matter to use the power of the computer itself to convert the code to ASCII, EBCDIC, Baudot (Murray), or whatever other code the builder may desire.

Thirty-three lines were used earlier to encode 32 discrete keyswitches. If the decoder is a 75154 and the MUX is a 74150, the 32 lines from the keyswitch matrix to the encoder will encode 256 switches (if you can find that many to encode!). The matrix is a 16 × 16, and the encoder capability is twice as much as the full ASCII character set. Extra keypads, cursor-controls, computer controls, and so on, can be encoded and the hexadecimal codes generated run from 00 to FF. figure 8 shows this circuit.

Computer Encoding

Keyboards may be encoded with a computer as well as with a hardware encoder. In this case we usually find ourselves restricted to 64 keys of encoding. This is caused by the 8-bit operation of our machines. We can use a PIA (Peripheral Interface Adapter) chip and tell 8 of the port lines they are outputs, and 8 more that they are inputs. With software, we then essentially duplicate the decoder function and drive each of the output lines low one-at-a-time. These correspond to the column lines of the hardware encoder. The keyswitches are placed in an 8 × 8 matrix so that the input lines (row lines) feed the input port. The keyswitch code generated in this fashion does not resemble hexadecimal in any sequence that you can dream up, but each key in the matrix will generate a discrete code. These discrete codes are then converted by software to the desired code. Figure 9 gives the basic circuit.

A full ASCII keyboard does not have 128 keys. The keys are made to do "double duty" with the Shift key. The Shift key on an 8 × 8 computer encoded matrix can be used to set one bit high and essentially double the number of codes generated. Special circuitry and software must be employed to handle the ASCII keyboard Control key. Thus, although the basic software encoder seems only capable of encoding either upper or lower case characters, it can, with some special circuitry encode almost all of the full 128 characters of the ASCII character set. We will have to give up a few of the ASCII codes, since two of the keys (Shift and Control) will need to perform special functions.

The basic problem of the computer encoded keyboard is that keyboards do not easily fit into a square matrix. Keyboards are rectangular and are usually 16 keys (or less) wide by 5 or 6 rows high. Consequently, if the foil traces must be cut to get the key-

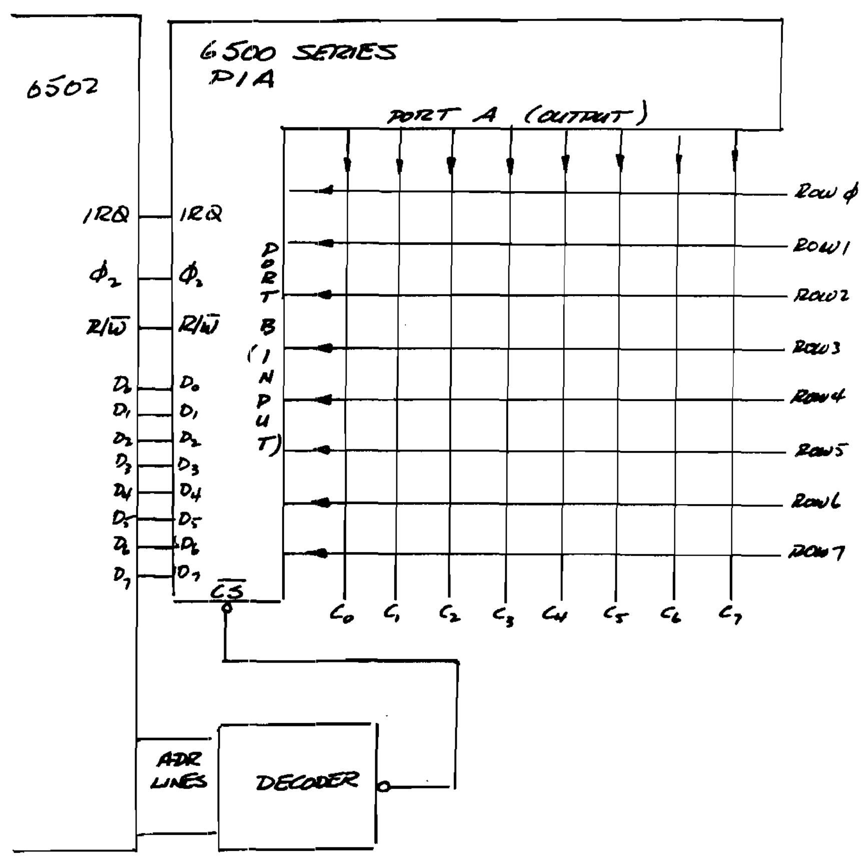


Figure 9. Basic circuit for matrix encoding using PIA chip.

switches into a matrix, the hardware 16 × 5 or 6 encoder will usually prove simpler to rewire than figuring out how to square things up and get the keys into an 8 × 8 matrix.

The Input Port

Once the keyboard has been encoded in hardware the builder must find a way to get its codes into the machine. One very definite advantage of the software encoded keyboard using a PIA chip is that this detail need not concern the builder.

To get into the machine, additional hardware is required. If a serial port is desired, this is usually handled with a UART. A parallel port is very easy to construct, and usually much less expensive to get into operation than the UART. (Figure 10 gives the circuitry.) The port is placed wherever the builder desires, with a decoder. The circuit of figure 10 will place the port at 7FF7, or about half-way through the address range. The keyboard code, which must be latched, is transferred to the accumulator with a LDA, Absolute Mode at address 7FF7. If the software is interrupt-driven, the IRQ latch may also be reset with this command. If the software is NMI-driven, then resetting of the IRQ latch is unnecessary. A STA, page zero or STA, Absolute will then place the keyboard code in memory and a Return From Interrupt can follow. The code can then be processed with a subroutine to convert it to whatever code you desire.

A bonus of the circuit of figure 10 is that not only one port is provided for, but 8 active lows are available from 7FF0-7FF7. This means that you can have 8 input or output ports at these locations. Or the signals generated by either reading or writing these memory locations can be used for handshakes or software generated turn-on or turn-off signals to the outside world. By placing a 1:16 decoder (74154) here instead of a 1:8 or 1:10 decoder, 16 ports, handshakes, etc. can be obtained.

Summary

The one basic circuit is quite flexible. It can be used to hardware encode almost any keyboard. Placing the keyboard in a matrix reduces the number or interconnecting wires needed between keyswitches and encoder circuitry. Since only one fundamental circuit was given here, and there are many, many ways to do the job, this article has, in reality, barely "scratched the surface" of encoding keyswitches.

George Young has been into computer electronics and programming for only three years. He has designed and built his own 6502-based homebrew computer in nine Phases, and this HB includes everything but floppies. Articles for publication are prepared on the text editing portion of this 6502 homebrew, and printing is via an interfaced Selectric I/O Writer.

The author's main interest is education He will teach *anything* to *anybody* who will take the time to listen to him or ask the questions and listen to the answers. He enjoys writing since it is an extension of his teaching (and enables him to have a larger number of students in his 'classroom').

MICRO

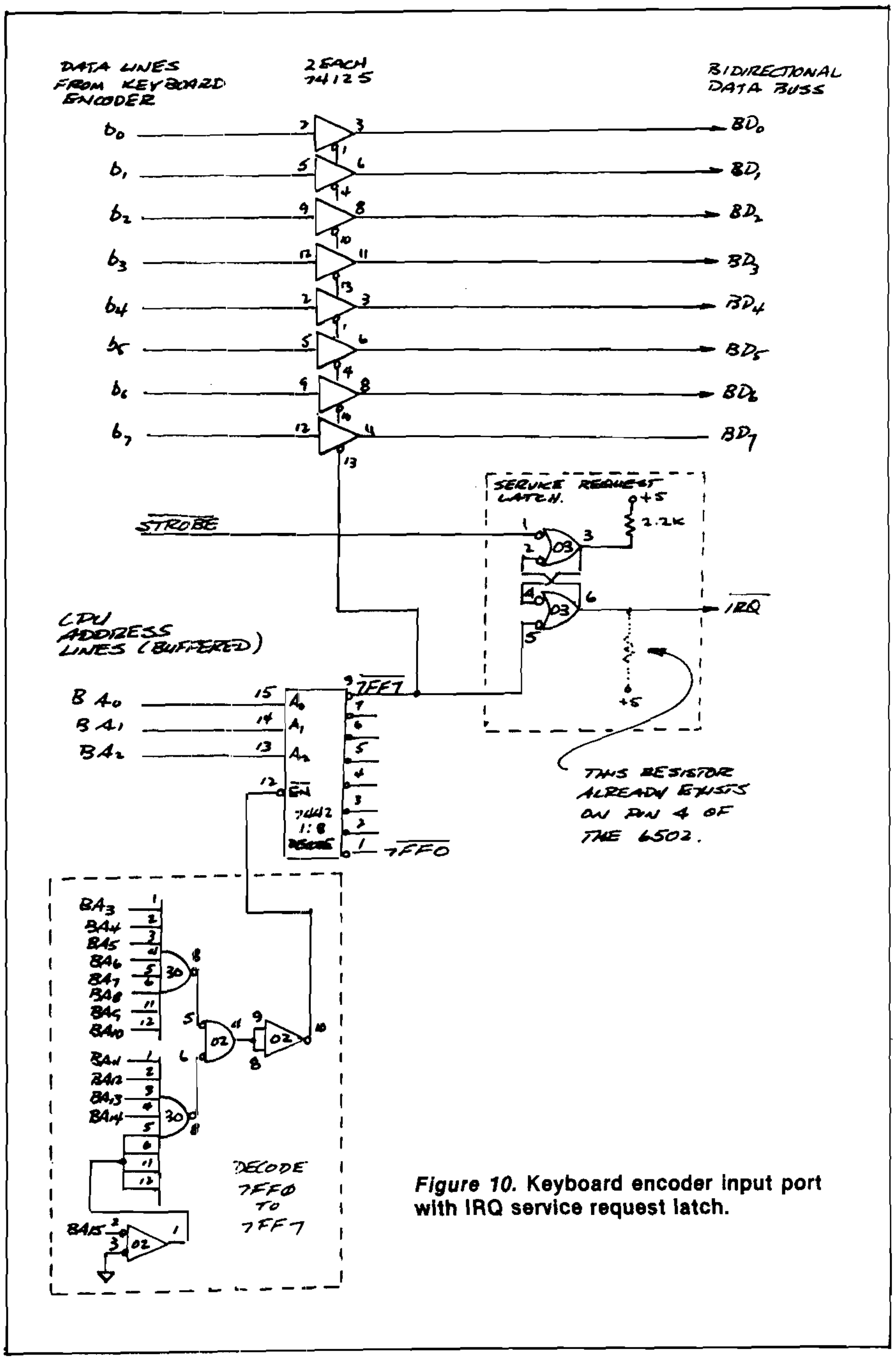
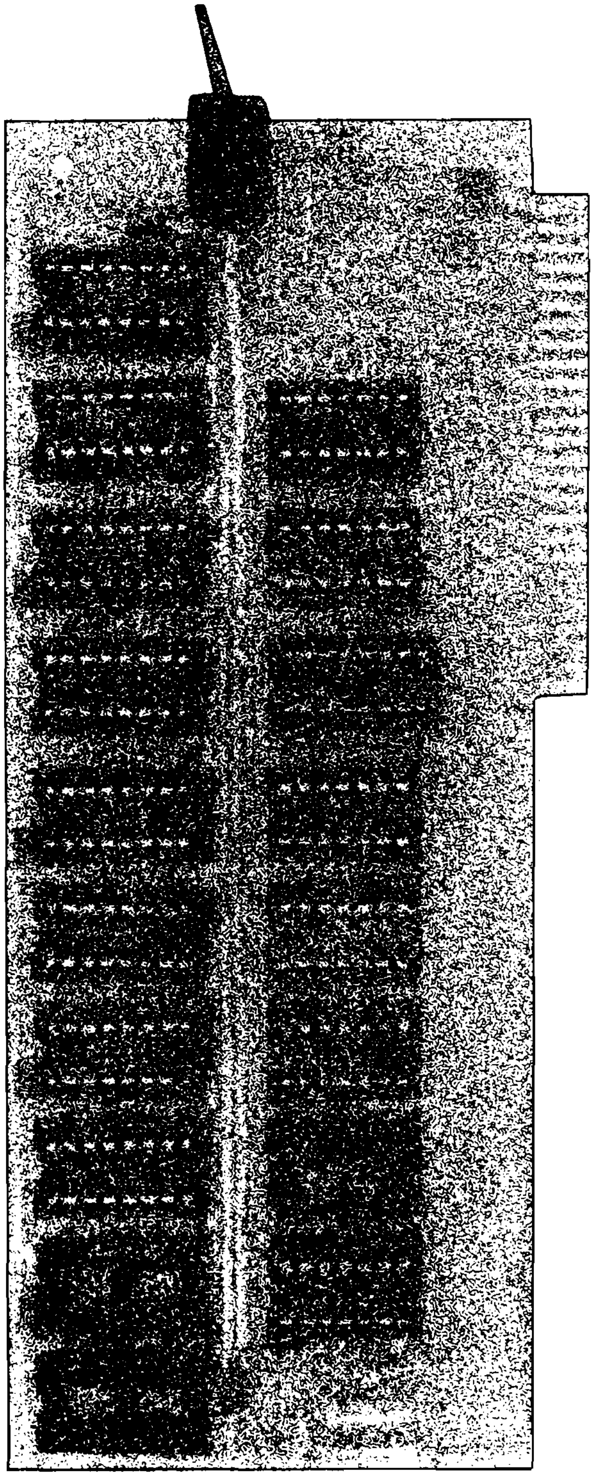


Figure 10. Keyboard encoder input port with IRQ service request latch.



Computer Camp

This summer, youngsters can again sign up for an overnight camp in Moodus, Connecticut, where the main activity will be COMPUTERS. This unique recreational and educational experience is directed by Dr. Michael Zabinski, Professor at Fairfield University. It is the original summer, overnight, computer camp currently offered in the USA.

The 1981 National Computer Camp will feature two action-packed one-week sessions: first session July 19-24; 2nd session July 26-31. The campers, ages 10-17, will enjoy small group instruction and mini and microcomputers for ample "hands-on" experience. Dr. Zabinski will be assisted by elementary and secondary school teachers.

The camp is for kids of all levels of experience, including no experience whatsoever. In addition to computers, the campers will enjoy the recreational facilities of the Grand View Lodge, which include swimming and volley ball.

For further information, contact Micahel Zabinski, Ph.D., at (203) 795-9069, or write to:

Computer Camp
Grand View Lodge, Box 22
Moodus, Connecticut 06469.

A Better Apple Search/Change

This improved version of the SEARCH/CHANGE program removes the length restriction on the CHANGE function.

J.D. Childress
5108 Springlake Way
Baltimore, Maryland 21212

Lately I've made much use of the SEARCH/CHANGE program published last January in MICRO (20:55) and have become quite impatient with its shortcomings. I first made some minor modifications but then decided the real need was to remove the length restriction on the CHANGE function. The complete program, with this modification, is given in the program listing. The discussion following assumes familiarity with the earlier paper.

An urgent warning is issued here. This program is like the girl with the curl: When it's good, it's very, very good but when it's bad, it's horrid. If it misfires, your program likely will be lost. The wise will always keep a current backup copy.

Minor Modifications

For those not wanting to do much retyping or needing only the range control feature (references here are to the line numbers in the SEARCH/CHANGE listing of the January paper): delete lines 63100, 63280, 63300, and 63310. These were written and kept in an abundance of caution. Change 63300 to 63320 after the GOSUB in line 63150. Add a line, say 63035, for input of BL, the beginning line for the change, and EL, the ending line; then insert OR LM < BL OR LM > EL in line 63350. The foregoing produces a shortening of the program, along with a capability for changing only a specified part of the program.

Another warning is in order here. The feature of listing the found lines works with a search for the LIST in line 63270 backwards from the end of program memory. If anything with a LIST in it is placed in higher line numbers, this search will be fouled up. And again note: somehow leading zeros in line 63270 can get removed; I suggest that a few, say 5, colons be inserted before INPUT.

Overview of Major Modification

Insertion of a CHANGE item longer or shorter than the SEARCH item requires that spaces either be added or deleted. This is accomplished by a shift of the program in memory and corresponding changes of all the next-address pointers. Needless to say, the part of memory space being used by SEARCH/CHANGE must not be jiggled, else its operation will be clobbered. So that the SEARCH/CHANGE program can remain fixed in memory and all the Applesoft operational pointers functional, spacers—colons—are added in line 62999.

Memory Move

The heart of this modification is the memory move call from Applesoft. (See lines 63360 and 63370 in the listing of this paper.) The memory move call given in *CONTACT* 5, 5 (June 79) works only for integer BASIC; a call to the Apple HOTLINE produced the information that the move call had to be routed through a short machine language routine. The routine supplied by Apple is the following: POKE 768,160: POKE 769,0: POKE 770,76: POKE 771,44: POKE 772,254. The corresponding call is then CALL 768. This location is $300-$304. I use that space for my SLOW LIST utility, so changed to location $342-$346 with no ill effects. (See line 63010.)

One block of memory cannot be moved into a second block overlapped by the first. This is because one byte would be moved into another before that byte's content had been read. Thus a two-step procedure is required. Line 63360 moves the memory block to the top of memory just below HIMEM and line 63370 moves it back to the desired end location.

Next-Address Pointers

Recall how Applesoft stores BASIC in memory. The end of each line is indicated by a zero byte. The next two bytes contains a pointer, low byte first, to the next line's first byte, the low byte of its next-address pointer. When a branch is executed, the program skips along these pointers from the first until the indicated address is found. If any next-address pointer points to a wrong address, all gang a-gley. So until all these pointers are put aright, the program being searched and changed is simply hidden from the operating program. Line 63160 POKEs the next-address pointer for line 63000 into the pointer location of line 1; after the dust settles, line 63230 restores the original pointer.

Other Matters

A search is made from the end of program memory to find two things: the location of the LIST in line 63310, and the location of the beginning of line 62999. This search is done by line 63020. Two numbers, 540 and 1730, are set for SEARCH/CHANGE exactly as written (lines 63000 and following) in the listing. The first number causes a skip from the end of the program to the immediate neighborhood of LIST, the second, a skip from LIST to near the end of line 62999. One effect of this search is a delay after the return following RUN 63000. A too-short delay should alert the user that line 62999 might not have enough colons for substantial changes. If the colons are depleted, line 63350 halts the change operation and prints a message to that effect.

```
62999  END :::::::::::::::::::::
     ::::::::::::::::::
63000  DIM SK(100),NT(100),L(100)
     :START = 256 *  PEEK (104) +
      PEEK (103):FINI = 256 *  PEEK
     (106) +  PEEK (105)
63010 HM = 256 *  PEEK (116) +  PEEK
     (115): POKE 834,160: POKE 83
     5,0: POKE 836,76: POKE 837,4
     4: POKE 838,254
63020  FOR WW = FINI - 540 TO STA
     RT STEP  - 1: IF X = 0 AND  PEEK
     (WW) = 188 THEN W = WW:X = 1
     :WW = WW - 1730
63030  IF 256 *  PEEK (WW + 1) +
      PEEK (WW) <  > 62999 THEN  NEXT

63040 NL = 256 *  PEEK (WW - 1) +
      PEEK (WW - 2):CO = NL - WW -
     10: HIMEM: HM - WW - 100
63050  IF 256 *  PEEK (START + 3)
      +  PEEK (START + 2) <  > 1 THEN
      PRINT "YOU MUST ENTER YOUR
     SEARCH ITEM AS LINE": PRINT
     "1 BEFORE YOU RUN 63000.": HIMEM:
     HM: END
63060  LIST 1,2: PRINT : PRINT "P
     LEASE VERIFY IF THE COMPUTER
      TAKES": PRINT "THIS AS YOU
     INTENDED. DO YOU WANT": INPUT
     "TO CONTINUE (YES OR NO)? ";
     Y$: PRINT : IF Y$ <  > "YES"
      THEN  HIMEM: HM: END
63070  PRINT "THE CHANGE ENTERED
     WILL BE MADE IN THE": PRINT
     "RANGE OF LINES CHOSEN. ENTE
     R": INPUT "    THE BEGINNING
      LINE   ";BL: INPUT "    THE
      ENDING LINE      ";EL: PRINT
63080 NF = 256 *  PEEK (START + 1
     ) +  PEEK (START)
63090  FOR I = 0 TO 255:SK(I) =  PEEK
     (START + 4 + I): IF SK(I) <
      > 0 THEN  NEXT
63100 N = I - 1
63110  IF SK(0) = 34 THEN  FOR I =
     1 TO N:SK(I - 1) = SK(I): NEXT
     :N = N - 1:SQ = 1
63120 M = START + N + 6 + SQ:INC =
     0:CH = 0: IF 256 *  PEEK (M +
     3) +  PEEK (M + 2) <  > 2 THEN
     CH = 1: GOTO 63170
63130  FOR I = 0 TO 255:NT(I) =  PEEK
     (M + 4 + I): IF NT(I) <  > 0
      THEN  NEXT
63140 NN = I - 1:ADD = NN - N:M =
     M + NN + 6:WW = WW - ADD(ADD
      < 0) + 8
63150  IF NT(0) = 34 THEN  FOR I =
     1 TO NN:NT(I - 1) = NT(I): NEXT
     :NN = NN - 1:ADD = ADD - 1
63160 X =  INT (NL / 256):Y = NL -
     256 * X: POKE START + 1,X: POKE
     START,Y
63170 LM = 256 *  PEEK (M + 3) +
      PEEK (M + 2):NA = 256 *  PEEK
     (M + 1) +  PEEK (M): IF LM >
      = 62999 THEN 63230
63180  FOR I = M + 4 TO M + 255: IF
      PEEK (I) <  > 0 AND  PEEK (
     I) <  > SK(0) AND  PEEK (I) <
      > 34 THEN  NEXT
63190  IF  PEEK (I) = 34 THEN SQ =
     SQ + 1 - 2 * (SQ = 1)
63200  IF  PEEK (I) = SK(0) AND S
     Q <  > 1 THEN  GOSUB 63330
63210  IF  PEEK (I) = 0 THEN NA =
     NA + INC:X =  INT (NA / 256)
     :Y = NA - 256 * X: POKE M +
     1,X: POKE M,Y:M = I + 1: GOTO
     63170
63220  NEXT
63230 X =  INT (NF / 256):Y = NF -
     256 * X: POKE START + 1,X: POKE
     START,Y
63240  PRINT : PRINT : PRINT "THE
      ITEM": PRINT "  ";: LIST 1:
      PRINT : PRINT "IS FOUND IN
     THE FOLLOWING LINES:": PRINT
     : IF L(1) = 0 THEN  PRINT "
          NONE.": HIMEM: HM: END
63250  FOR I = 1 TO N: PRINT L(I)
     ,: NEXT : PRINT
63260  PRINT : INPUT " DO YOU WAN
     T THESE LINES LISTED (YES OR
      NO)? ";Y$: IF Y$ = "NO" THEN
      HIMEM: HM: END
63270  PRINT : PRINT "THERE WILL
     BE A WAIT AFTER EACH LINE": PRINT
     "UNTIL YOU HIT RETURN TO CON
     TINUE.": PRINT
63280  FOR I = 1 TO K: IF L(I) =
     L(I - 1) THEN 63320
63290 L$ = "0000" +  STR$ (L(I)):
     L$ =  RIGHT$ (L$,5)
63300  FOR J = 1 TO 5: POKE W + J
     ,48 +  VAL ( MID$ (L$,J,1)):
      NEXT
63310  LIST 00050:::::::::: INPUT
      "";Y$
63320  NEXT : HIMEM: HM: END
63330  FOR J = 0 TO N: IF  PEEK (
     I + J) <  > SK(J) THEN  RETURN

63340  NEXT
63350  IF CO - INC < ADD AND CO% =
     0 THEN CH = 1:CO% = 1: PRINT
     "THE SUPPLY OF COLONS IN LIN
     E 62999 IS": PRINT "DEPLETED
     .  THE CHANGE HAS BEEN MADE"
     : PRINT "THROUGH PART OR ALL
      OF LINE ";LM: PRINT
```

```
63360 K = K + 1:L(K) = LM: IF CH <
      > 0 OR LM < BL OR LM > EL THEN
      RETURN
63370 ZS = I + N:X =  INT (ZS / 2
      56):Y = ZS - 256 * X: POKE 6
      1,X: POKE 60,Y:X =  INT (WW /
      256):Y = WW - 256 * X: POKE
      63,X: POKE 62,Y:ZH = HM - 10
      0 - WW + ZS:X =  INT (ZH / 2
      56):Y = ZH - 256 * X: POKE 6
      7,X: POKE 66,Y: CALL 834
63380  POKE 61,X: POKE 60,Y:X =  INT
      ((HM - 100) / 256):Y = HM -
      100 - 256 * X: POKE 63,X: POKE
      62,Y:ZS = I + NN:X =  INT (Z
      S / 256):Y = ZS - 256 * X: POKE
      67,X: POKE 66,Y: CALL 834
63390 WW = WW + ADD:INC = INC + A
      DD
63400  FOR J = 0 TO NN: POKE I +
      J,NT(J): NEXT :I = I + NN
63410  RETURN
```

The perceptive reader will note a number of small numbers in various lines. These are finagle factors adjusted (but probably not optimized) to make the program run. For example, the 10 in line 63040 prevents the appearance of multiple line 62999's for a change item shorter than the search item.

I have also changed the choices for search and the way they work. Experience has shown no need to search both inside and outside strings at the same time; it's an either/or situation. To eliminate a needless question/answer routine, the program now works as follows: the SEARCH/CHANGE is made outside strings only, unless the first character of line 1 is the quotation mark. In that case, the SEARCH/CHANGE is made inside strings only. For example,

1 TOTAL

would search for TOTAL outside strings but

1 "TOTAL

would search for TOTAL inside strings.

A quotation mark can be used with line 2 in a similar way to sneak "forbidden" words past the interpreter. This should be used with care in changes outside strings; the interpreter has a way of exacting its revenge on sneaky things.

Operation

Except for the search mode selection change, operation of the better SEARCH/CHANGE is essentially the same as the original. Append SEARCH/CHANGE to the program to be searched. Enter the search item as line 1 and the replacement item as line 2. Note that anything that will list as line 1 (or line 2) can be entered. Execute with a RUN 63000.

As mentioned earlier, things can go wrong. If the worst happens, try entering a new line, or deleting a line, or both. That sometimes will save part, or almost all of the program. A sensible precaution is to check line 62999 often and keep it well-stocked with colons.

Another good idea is to use a SLOW LIST utility with SEARCH/CHANGE. (I recommend the one supplied with S-C ASSEMBLER II.) Then if a LIST command produces endless junk, the listing can be aborted without the additional hazard of a RESET.

Another idea is to know your HIMEM. If something goes wrong, it's possible for the HIMEM setting to be ratcheted down to a low value.

A problem I encountered on occasion and, I trust, eliminated, is the DOS 3.2 renumbering program (which I keep in memory while programming) or the DOS buffers being clobbered, probably by an occasional encroachment into the space above HIMEM.

MICRO

Vectors and the Challenger 1P

Vector is one more computer buzzword usually found in conjunction with confusion. This article will try to clear up what vectors are and will show how to use them effectively.

Mike Bassman
39-65 52nd Street
Woodside, New York 11377

Computers have subroutines for every command and for every other necessary function. Computers also need places in memory to look up the address of a particular subroutine. A vector is a place in memory in which the computer finds the address of a subroutine. The vector will consist of two bytes containing any address from $0000 to $FFFF in low-byte, high-byte format.

I'll try to clear this up with an example. Let's say you type in a SAVE command. The C1P must know where to go to find the SAVE subroutine. The C1P looks at the SAVE vector, which is at $220 and $221 (544 and 545 decimal), and in it finds $96 and $FF, which is $FF96, the address of the SAVE routine.

What good are vectors? They are used if you want to add to BASIC or change it or any of its commands. Vectors have been used to create shorthand (see MICRO 24:25), do a true backspace, and ensure program security. Let's take program security as an example. A good way to prevent copying of tapes is to disable the SAVE command. We want the SAVE command to coldstart BASIC if someone tries to save a program. The SAVE vector must be changed from pointing at the SAVE routine at $FF96 to the coldstart address at $BD11. Therefore, the SAVE vector at 544 and 545 must point at $BD11. These two poke commands will take care of it:

```
POKE 544,17
POKE 545,189
```

Now, if you type SAVE, the C1P will respond with "MEMORY SIZE?". Although the C1P has vectors for every command, a large portion of them are in ROM rather than in RAM. Since only RAM and not ROM can be changed, only those vectors residing in RAM can be used. Vectors in RAM, their addresses, and the address that they point at initially are listed below.

Vector	Address Hex	Address Decimal	Initial Value Hex	Initial Value Decimal
SAVE	220,221	544,545	FF96	65430
LOAD	21E,21F	542,543	FF8B	65419
CTRL C	21C,21D	540,541	FF9B	65435
OUTPUT	21A,21B	538,539	FF69	65385
INPUT	218,219	536,537	FFBA	65466
WARM START	001,002	001,002	A274	41588

The first two vectors, the SAVE vector and the LOAD vector are used whenever the SAVE and LOAD commands are executed. The Control-C vector is somewhat more subtle. For every line of BASIC executed, BASIC checks the keyboard for a Control-C. If one has been typed, program execution ends. This vector is the one that is used for every line executed. Therefore, we can use this vector, if our function is one that would be executed once for every line, just as we would use a program tracer. The output vector is used each time BASIC wants to type a character. The input vector is used each time BASIC wants to input a character.

Let's try a more involved example. This time, we'll change the cursor from the underline to, say, a tank character. The program will require use of the input vector, because we will have to change the cursor each time a character is inputted. The program will be in the free part of page 2, starting at $222 (546 decimal). BASIC stores the cursor location at $200 (513 decimal). This is used as an index from screen location $D300 (54016 decimal). Since BASIC resets the input vector after carriage return is hit, we will set the input vector to $222 after every character. The program follows.

Using these techniques, Ohio Scientific C1P users can customize BASIC to their convenience.

```
                    BASSMAN:VECTORS AND THE CHALLENGER 1P
0222                        ORG   $0222
0222 48                     PHA             SAVE ACCUMULATOR
0223 8A                     TXA             PUT X IN A
0224 48                     PHA              AND SAVE
0225 AE 00 02               LDX   $0200     GET CURSOR LOCATION
0228 A9 FA                  LDA   #$FA      GET TANK CHARACTER
022A 9D 00 D3               STA   $D300,X   STORE WITH INDEX
022D 68                     PLA             GET X FROM STACK
022E AA                     TAX             AND PUT INTO X
022F A9 22                  LDA   #$22      LOW BYTE--INPUT VECTOR
0231 8D 18 02               STA   $0218     AND SAVE
0234 A9 02                  LDA   #$02      HIGH BYTE--INPUT VECTOR
0236 8D 19 02               STA   $0219     AND SAVE
0239 68                     PLA             GET A
023A 4C BA FF               JMP   $FFBA     AND RETURN
```

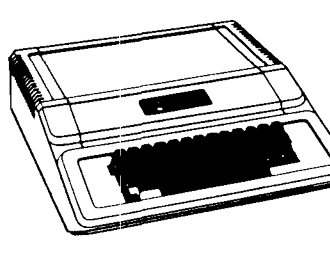
MICRO

THE STAR MODEM

From Livermore Data Systems

RS232 MODEM	SALE $135
IEEE 488 MODEM	SALE $245
RS232 CCITT	$170
IEEE 488 CCITT	$280

STAR Modem is the price performance leader with a full 2 YEAR FACTORY WARRANTY.



WE CARRY THE BLACK APPLE

BELL & HOWELL

MADE EXCLUSIVELY FOR BELL & HOWELL BY

apple computer inc.

EBS Business System for PET/CBM — $795

Extremely comprehensive package for small business. Fully integrated inventory and accounts receivable system including invoices, packing slips, mail labels, statements, bank deposit slips, and 17 reports. Allows any of 10 standard letters to be merged with customer record info on either a selective or complete file basis. Demo disk and system description available for $3.00.

OZZ Data Base System for CBM 8032 — $335

Flexible file handling and report writing package for such applications as: Inventory Control; Management Information; Mailing List; Scheduling; Medical Record Keeping; Accounting.

KMMM Pascal for PET — $75

Subset of standard Pascal with true machine language translator for faster execution. 16K with tape or disk.

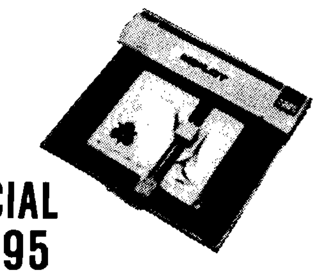
EARL for PET (disk file based) — $65

Editor, Assembler, Relocater, Linker to generate relocatable object code.

fullFORTH+ for PET/CBM — $65

A full-featured FORTH with extensions conforming to Forth Interest Group standards. Includes assembler, string processing capabilities, disk virtual memory multiple dimensioned arrays, floating point and integer processing.

MIPLOT Intelligent Plotter
by Watanabe Instruments (Digiplot)



SPECIAL $1095

- Incorporates all intelligent functions required for producing graphs and drawings including 8 vector and 4 character commands.
- Distance accuracy within 1%, repetition accuracy within .01 inches, programmable step size .004 inches, internal interpolation in .002 steps.
- Solid and broken line types can be specified.
- Character generator for letters, numerals and symbols. Characters can be enlarged and rotated in four orientations.
- Coordinate axes can be drawn by simply specifying the graduation interval and number of repetitions.
- Self test mode automatically draws complete test pattern.
- Printer mode outputs character data in 16 sizes with 4 orientations.
- Connection to any microcomputer using parallel 7-bit ASCII code.
- Can use simple cable to parallel port, or special interface to IEEE or other.
- Modular control circuit and mechanical construction.
- Uses any hard fiber-tip pen.
- Uses 11 x 17 paper.

Includes power supply, I/O connector, 2 pens, 50 sheets of paper, and complete manual.

6502	7.45	10/6.95	50/6.55	100/6.15
6502A	8.40	10/7.95	50/7.35	100/6.90
6520 PIA	5.15	10/4.90	50/4.45	100/4.15
6522 VIA	6.45	10/6.10	50/5.75	100/5.45
6532	7.90	10/7.40	50/7.00	100/6.60
2114-L450		3.95	20/3.75	100/3.45
2114-L300		4.45	20/4.25	100/3.95
2716 EPROM		13.45	5/12.75	10/11.85
2732 EPROM (TMS2532)				29.50
4116-200 ns RAM (NEC)				**8 for 35.00**
S-100 Wire Wrap $2.65				Solder Tail $2.15

CASSETTES – AGFA PE-611 PREMIUM

High output, low noise, 5 screw housing, labels.

C-10	10/5.65	50/25.00	100/48.00
C-30	10/7.30	50/34.00	100/66.00

All other lengths available. Write for price list.

ATARI 800 $777

All Atari Modules 20% OFF

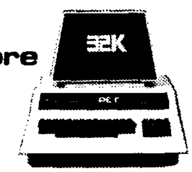
SPECIAL–purchase ATARI 800, receive extra 8K memory FREE.

EDUCATIONAL PLAN - buy 2 ATARI Computers, receive 1 ATARI 400 FREE!

DISKS
(write for quantity prices)

SCOTCH (3M) 8"	10/3.10	50/2.85	100/2.75
SCOTCH (3M) 5"	10/2.95	50/2.80	100/2.70
Maxell 5" Double Dens	10/4.25	50/4.10	100/3.95
Maxell 8" Double Dens.	10/4.65	50/4.50	100/4.35
Verbatim 5"	10/2.40	50/2.35	100/2.30
(add .75 for plastic storage box for Verbatim 5")			
Verbatim 8" Dbl. Dens.	10/3.35	50/3.25	100/3.15
BASF 5"	10/2.60	20/2.50	100/2.40
BASF 8"	10/2.40	20/2.35	100/2.30
Diskette Storage Pages			10 for 3.95
Disk Library Cases		8" - 2.85	5" - 2.15

commodore



CBM-PET SPECIALS

FREE Up to $235 free merchandise with purchase of one of following CBM-PET items!

		FREE
8032 32K - 80 Column CRT	$1795	235
8016 16K - 80 Column CRT	$1495	205
8050 Dual Disk Drive-1,020,000 Bytes	$1795	235
CBM Modem - IEEE Interface	$395	50
CBM Voice Synthesizer	$395	50
8N Full size graphics keyboard	$795	100
16K Full Size Graphics or Business Keyboard	$995	150
32K Full Size Graphics or Business Keyboard	$1295	205
2040 Dual Disk Drive - 343,000 bytes	$1295	205
2022 Tractor Feed Printer	$795	100
C2N External Cassette Deck	$95	12
Used PETs (8, 16, and 32K)	CALL	

WRITE FOR SYSTEM PRICES

****EDUCATIONAL DISCOUNTS****

Buy 2 PET Computers, get 1 FREE

CBM Full Size Graphics Keyboard	$ 74
WordPro III - 32K CBM, disk, printer	$170
WordPro IV - 8032, disk, printer	$255
VISICALC for PET, ATARI	$170
BPI General Ledger, A/P, A/R for PET	$270
Programmers Toolkit - PET ROM Utilities	$34.90
PET Spacemaker Switch	$24.90
Dust Cover for PET	$6.90
IEEE-Parallel Printer Interface for PET	$65.00
IEEE-RS232 Printer Interface for PET	$149.00

EPSON MX-80 Printer	$545
C. Itoh Daisy Wheel Printer	$1575
Centronics 737 Proportional Spacing Printer	$795
NEC Spinwriter - parallel	$2450
XYMEC HI-Q 1000 Intelligent Daisy Wheel	$1975
Leedex Video 100 12" Monitor	$129
ZENITH DATA SYSTEMS	
Zenith Z19 Terminal (factory asm.)	$735
Zenith Z89 with 48K	$2150
SYM-1	$209
SYM BAS-1 BASIC or RAE-1/2 Assembler	$ 85
KTM-2/80 Synertek Video Board	$349
KIM-1 (add $34 for power supply)	$159
Seawell Motherboard - 4K RAM	$195
Seawell 16K Static RAM - KIM, SYM, AIM	$320
KL-4M Four Voice Music Board and Visible Music Monitor (4 Voice) for PET	$59.90

FLEXFILE Database-Report Writer by Michael Riley
Flexible file handler for PET/CBM

MICRO-REVERSI for PET by Michael Riley — $9.95
Machine language version—you can't win at Level 5.

PAPER-MATE 60 Command PET Word Processor $29.95
Full-featured version by Michael Riley

A P Products 15% OFF
A P Hobby-Blox 15% OFF

ALL BOOK and SOFTWARE PRICES DISCOUNTED

The 8086 Book (Osborne)	$12.75
Z8000 Assembly Language Programming	$10.60
PET Personal Computer Guide (Osborne)	$12.75
PET and the IEEE-488 Bus (Osborne)	$12.75
6502 Assembly Language (Osborne)	$ 9.90
Programming the 6502 (Zaks)	$10.45
6502 Applications Book (Zaks)	$10.45
6502 Software Cookbook (Scelbi)	$ 9.45
CP/M Handbook (w/ MP/M) Zaks	$11.85

115 E. Stump Road
Montgomeryville, PA 18936 — 215-699-5826 — **A B Computers**

WRITE FOR CATALOG.

Add $1.25 per order for shipping. We pay balance of UPS surface charges on all prepaid orders. Prices listed are on cash discount basis. Regular prices slightly higher.

PET Symbolic Disassembler

Most disassemblers output only absolute addresses. This symbolic disassembler generates labels and symbols for these addresses, making the disassembled program much more understandable.

Werner Kolbe
Hardstrasse 77
CH5432 Neuenhof
Switzerland

If you want to understand how programs written in machine code work, you need a clear and comprehensive listing. For this purpose I developed a disassembler (listing 1) which produces a listing nearly like one from an assembler—only the remarks are missing. Because I used the program mainly to get a complete listing of the PET ROM routines, the program also contains a short machine routine to overcome the PEEK limitation. The program automatically creates names for all memory locations used by the machine program. It also calculates the absolute addresses of all branches and assigns labels automatically. Listing 2 shows an example of a listing produced by the disassembler.

Using the Disassembler

Because the program is very simple to use, explanations are not included, in order to save space. The program first asks for the output device number, which should be 3 for the screen or 4 for the printer. Then it asks for the start address (origin) of the program to be disassembled. The address has to be entered in hexadecimal without a leading $ sign. Leading zeros are not necessary. Program operation can be stopped by pressing the SPACE key and continued by pressing the key a second time. By pressing the HOME key, the program jumps back to the start, asking again for the output device and the start address. However, it still holds all names created thus far. To get a complete listing it is necessary to go through the machine code a second time, because all labels created by backward jumps can only be resolved in a second pass. Tables of the labels and variable names are printed when, instead of a start address, the following keywords are entered:

- PM all labels are listed
- PL only the labels created by a branch (they start with L) are listed
- PJ only the labels created by a JMP or JSR are listed (they start with J)
- PZ a listing of all zero-page memory places used is given (their names start with Z)
- PW a listing of all memory places above page zero used by the program is given (names starting with W).

By entering "ENTRY" instead of a start address, you can name your own *variables*. This must be done before the program has created a name, because a variable cannot be defined twice. Provision for entering your own *label* names has not been made. You can leave the "ENTRY"-mode by entering END as a name. Addresses in "ENTRY"-mode must be hexadecimal without a leading $. The program itself occupies less than 4K bytes, but to provide enough space for all addresses and names that have to be stored, you should have at least about 16K bytes of memory available.

Program Description

Line No.	Description
5	determines the dimension of arrays L,L$, and P,P$ which contain the labels (L) and the page>0 (P) names. If you have a new ROM and enough memory you could set DM and PM to higher values.
10...15	loads a short machine routine which overcomes the PEEK limitation in the old ROMs.
20	dimensions the arrays. M$ = mnemonic code, Z% = zero-page addresses, Z$ = zero-page names, L, P (see above)
25	the addresses 0 are assigned to provide the headings for the listings.
35	reads mnemonic code, including addressing index.
45	assigns output device.
50...72	FL is a "text"-flag. If FL = 0, then text mode is assumed until a zero is found. The "address" in E$ is investigated in the following lines to detect the keywords.
75...90	subroutine 280 changes E$ into a number in E. P is a pointer to the current address. Subroutine 325 searches for a label under the address in P. S = 1 tells the subroutine not to create a label if none is found. Subroutine 300 performs a decimal to hexadecimal conversion. Subroutine 295 performs an "E = PEEK(P)" and the hexadecimal of E is returned in E$.

92 separates mnemonic code and addressing index.

95 jumps to the subroutines according to the addressing index.

105 the start/stop feature using the SPACE key.

130 prints the mnemonic code at the correct place. As TAB does not work on my printer, a counter K and the SPC function is used.

135...160 handles implied addressing and text.

165 immediate addressing mode.

170 zero-page addressing mode.

175 fetch operand; print it and mnemonic; look for entry in the address table and create name, if not found (subroutine 380); print name plus addressing string.

180 zero-page, X addressing mode.

185 zero-page, Y addressing mode.

190 absolute addressing.

195...205 same task as 175, only for absolute addressing. If no new names can be created because the arrays P,P$ are full, the address is printed in hexadecimal.

210 absolute, X addressing.

215 absolute, Y addressing.

220...230 relative addressing. If the label was created anew and the branch is backward, the address is printed in hexadecimal. The same occurs if the arrays L,L$ are full.

235 (indirect), X addressing.

240 (indirect), Y addressing.

250 absolute indirect addressing; the operand is printed in hexadecimal. It seemed to be unnecessary to create a table of two-byte pointer names only for this seldom-used addressing mode.

260 accumulator addressing.

265...275 JMP and JSR absolute. Similar task as 220...230.

280...285 converts hexidecimal in E$ to decimal in E.

290...320 increments code pointer P, PEEKS the memory location P, and returns the value in E and ES.

325...375 looks for the address E in the array L; if found, returns corresponding name in L$; if not, creates a new name and sorts the table. The new name starts with "L" or "J" and continues with a current number which is the "end of table L" pointer LL. A binary search is executed to improve program speed.

380...415 same as above but for zero-page addresses.

420...465 same as above but for addresses above page zero.

470...540 mnemonic code and assigned addressing index.

545...605 prints address tables and enables entry of addresses and corresponding names.

Werner Kolbe is a German computer enthusiast who has been living in Switzerland since 1977, working as an electrical engineer in the field of power system protection. In 1978 he bought one of the first PETs available in Europe. With some programming experience in FORTRAN gained on a large IBM computer, using his PET is really enjoyable.

DISASSEMBLER

```
5 DM=255:PM=50
10 FORI=1011TO1017:READA:POKEI,A:NEXT
15 DATA173,0,0,141,250,3,96
20 DIMM$(255),L$(DM),L(DM),Z%(255),Z$(255),P(PM),P$(PM)
25 DM=DM-1:PM=PM-1:Z$(0)="ZERO":L$(0)="LABEL":P$(0)="PAGE >0"
30 FORI=0TO255:READM$(I):NEXT
45 CLOSE1:INPUT"OUTPUT DEV. #";D:PRINT:OPEN1,D,1
50 FL=1:INPUT" ORG";E$:PRINT:IFE$="PM"THENV$="":GOTO545
55 IFE$="PL"THENV$="L":GOTO545
60 IFE$="PJ"THENV$="J":GOTO545
65 IFE$="PZ"THEN560
70 IFE$="PW"THEN570
72 IFE$="ENTRY"THEN600
75 GOSUB280:P=E-1
80 P=P+1:E=P:S=1:GOSUB325:IFL$<>" "THENFL=1
85 GOSUB300
90 PRINT#1,SPC(5-LEN(E$))E$;:GOSUB295:PRINT#1," "E$;:K=5
92 M$=LEFT$(M$(E),3):B=VAL(MID$(M$(E),4))
95 ONB*FL+1GOSUB135,165,170,180,185,190,210,215,220,235,240,250,260,265
105 GETE$:IFE$<>""THENV$=" ":GOSUB115
110 GOTO80
115 IFE$=V$THENGETE$:V$="":GOTO115
120 IFE$="S"THENPRINT#1:GOTO45
125 RETURN
```

```
130 PRINT#1," "E$SPC(K)L$SPC(7-LEN(L$))M$" ";:RETURN
135 IFFL=1ANDM$<>"?"THENPRINT#1,SPC(8)L$SPC(7-LEN(L$))M$:RETURN
140 FL=0:IFM$="BRK"THENFL=1:PRINT:RETURN
145 PRINT#1,SPC(15)"? : "CHR$(34);:IFE>30ANDE<128THENPRINT#1,CHR$(E)
150 IFE<30THENPRINT#1,CHR$(E+64)
155 IFE>127THENPRINT#1,CHR$(E-128)
160 RETURN
165 GOSUB290:GOSUB130:PRINT#1,"="E$:RETURN
170 V$=""
175 GOSUB290:GOSUB130:GOSUB380:PRINT#1,Z$V$:RETURN
180 V$=",X":GOTO175
185 V$=",Y":GOTO175
190 V$=""
195 GOSUB290:PRINT#1," "E$;:H$=E$:GOSUB290:K=2
200 GOSUB130:E$=E$+H$:GOSUB280:GOSUB420:IFP$<>""THENPRINT#1,P$V$:RETURN
205 PRINT#1,E$V$:RETURN
210 V$=",X":GOTO195
215 V$=",Y":GOTO195
220 GOSUB290:A1=E:GOSUB130:E=A1+P+1:IFA1>127THENE=P-255+A1
225 V$="L":S=0:GOSUB325:IFBTHENPRINT#1,L$:RETURN
230 GOSUB300:PRINT#1,E$:RETURN
235 V$=",X)":GOTO245
240 V$="),Y"
245 GOSUB290:GOSUB130:GOSUB380:PRINT#1,"("Z$V$:RETURN
250 GOSUB290:PRINT#1," "E$;:H$=E$:GOSUB290:K=2:GOSUB130
255 PRINT#1,"("E$H$")":RETURN
260 PRINT#1,SPC(8)L$SPC(7-LEN(L$))M$"A":RETURN
265 GOSUB290:PRINT#1," "E$;:H$=E$:GOSUB290:K=2:GOSUB130:E$=E$+H$:GOSUB280
270 V$="J":S=0:GOSUB325:IFL$<>" "THENPRINT#1,L$:RETURN
275 PRINT#1,E$:RETURN
280 E=0:FORI=1TOLEN(E$):B=ASC(MID$(E$,I,1))-48:IFB>9THENB=B-7
285 E=E*16+B:NEXT:RETURN
290 P=P+1
295 B=INT(P/256):POKE1013,B:B=P-B*256:POKE1012,B:SYS1011:E=PEEK(1018)
300 B=E:E$=""
305 H=INT(B/16):B=INT(B-16*H):B$=CHR$(B+48):IFB>9THENB$=CHR$(55+B)
310 E$=B$+E$:IFH>=1THENB=H:GOTO305
315 IFLEN(E$)<2THENE$="0"+E$
320 RETURN
325 B=-1:H=LL+1
330 I=INT((H+B)/2):IFL(I)=ETHENB=1:L$=L$(I):RETURN
335 IFL(I)>ETHENH=I:GOTO345
340 B=I
345 IFABS(H-B)>1THEN330
350 IFSOR(LL>DM)THENB=0:L$=" ":RETURN
355 LL=LL+1:IFL(I)<ETHENI=I+1
360 FORB=LLTOI+1STEP-1:L(B)=L(B-1):L$(B)=L$(B-1):NEXT
365 L(I)=E:L$(I)=V$+MID$(STR$(LL),2)
370 B=0:L$=L$(I):IFE>PTHENB=1
375 RETURN
380 B=-1:H=ZZ+1
385 I=INT((H+B)/2):IFZ%(I)=ETHENZ$=Z$(I):RETURN
390 IFZ%(I)>ETHENH=I:GOTO400
395 B=I
400 IFABS(H-B)>1THEN385
405 ZZ=ZZ+1:IFZ%(I)<ETHENI=I+1
410 FORB=ZZTOI+1STEP-1:Z%(B)=Z%(B-1):Z$(B)=Z$(B-1):NEXT:Z%(I)=E
412 IFS=2THENZ$(I)=V$:RETURN
415 Z$(I)="Z"+MID$(STR$(ZZ),2):Z$=Z$(I):RETURN
420 IFE<256THENGOSUB380:P$=Z$:RETURN
425 B=-1:H=PP+1
430 I=INT((H+B)/2):IFP(I)=ETHENP$=P$(I):RETURN
435 IFP(I)>ETHENH=I:GOTO0445
440 B=I
```

(continued)

```
445 IFABS(H-B)>1THEN430
450 IFPP>PMTHENP$="":RETURN
455 PP=PP+1:IFP(I)<ETHENI=I+1
460 FORB=PPTOI+1STEP-1:P(B)=P(B-1):P$(B)=P$(B-1):NEXT:P(I)=E
462 IFS=2THENP$(I)=V$:RETURN
465 P$(I)="W"+MID$(STR$(PP),2):P$=P$(I):RETURN
470 DATABRK,ORA9,?,?,?,ORA2,ASL2,?,PHP,ORA1,ASL12,?,?,ORA5,ASL5,?,BPL8,ORA10
475 DATA?,?,?,ORA3,ASL3,?,CLC,ORA7,?,?,?,ORA6,ASL6,?,JSR13,AND9,?
480 DATA?,BIT2,AND2,ROL2,?,PLP,AND1,ROL12,?,BIT5,AND5,ROL5,?,BMI8,AND10,?,?,?
485 DATAAND3,ROL3,?,SEC,AND7,?,?,?,AND6,ROL6,?,RTI,EOR9,?,?,?,EOR2,LSR2
490 DATA?,PHA,EOR1,LSR12,?,JMP13,EOR5,LSR5,?,BVC8,EOR10,?,?,?,EOR3,LSR3,?
495 DATACLI,EOR7,?,?,?,EOR6,LSR6,?,RTS,ADC9,?,?,?,ADC2,ROR2,?,PLA,ADC1
500 DATAROR12,?,JMP11,ADC5,ROR5,?,BVS8,ADC10,?,?,?,ADC3,ROR3,?,SEI,ADC7,?,?
505 DATA?,ADC6,ROR6,?,?,STA9,?,?,STY2,STA2,STX2,?,DEY,?,TXA,?,STY5,STA5
510 DATASTX5,?,BCC8,STA10,?,?,STY3,STA3,STX3,?,TYA,STA7,TXS,?,?,STA6,?,?
515 DATALDY1,LDA9,LDX1,?,LDY2,LDA2,LDX2,?,TAY,LDA1,TAX,?,LDY5,LDA5,LDX5,?
520 DATABCS8,LDA10,?,?,LDY3,LDA3,LDX4,?,CLV,LDA7,TSX,?,LDY6,LDA6,LDX7,?
525 DATACPY1,CMP9,?,?,CPY2,CMP2,DEC2,?,INY,CMP1,DEX,?,CPY5,CMP5,DEC5,?
530 DATABNE8,CMP10,?,?,?,CMP3,DEC3,?,CLD,CMP7,?,?,?,CMP6,DEC6,?,CPX1,SBC9
535 DATA?,?,CPX2,SBC2,INC2,?,INX,SBC1,NOP,?,CPX5,SBC5,INC5,?,BEQ8,SBC10,?,?
540 DATA?,SBC3,INC3,?,SED,SBC7,?,?,?,SBC6,INC6,?
545 FORI=0TOLL:IFV$<>""THENIFLEFT$(L$(I),1)<>V$THEN555
550 E=L(I):GOSUB300:PRINT#1,L$(I)SPC(8-LEN(L$(I)))E$SPC(8-LEN(E$));
555 NEXT:PRINT#1:GOTO45
560 FORI=0TOZZ:E=Z%(I):GOSUB300
565 PRINT#1,Z$(I)SPC(8-LEN(Z$(I)))E$SPC(8-LEN(E$));:GOTO555
570 FORI=0TOPP:E=P(I):GOSUB300
575 PRINT#1,P$(I)SPC(8-LEN(P$(I)))E$SPC(8-LEN(E$));:GOTO555
600 S=2:INPUT"NAME,ADRESS";V$,E$:IFV$<>"END"THENGOSUB280:GOSUB420:GOTO600
605 GOTO45
1000 FORI=14TO33:PRINTICHR$(34)CHR$(I)CHR$(34);:NEXT
READY.
```

MICRO

AIM 65 File Operations

This, the third part of a series, builds upon previous articles to arrive at an AIM 65 text file processing system with BASIC.

Christopher J. Flynn
2601 Claxton Drive
Herndon, Virginia 22070

In two previous articles (26:61) and (30:65), we described text file input and output subroutines. These subroutines, when used with BASIC, added significant file handling capabilities to the AIM 65. We will now take these capabilities a step further, showing how BASIC can make effective use of the AIM 65's built-in dual cassette interface. We will remove our earlier restriction on the concurrent use of input and output files. The result, then, will be a general-purpose file system for processing AIM 65 text files with BASIC.

AIM 65 Hardware

The term "dual cassette interface" may be a little misleading to some. The AIM 65 has, in fact, one audio input channel, one audio output channel, and two motor control circuits.

Now, to use two tape recorders with the AIM 65, we dedicate one of them as a playback-only drive and the other as a record-only drive. In our system, we use drive 1 for playback and drive 2 for recording. The motor control circuits are connected to the appropriate tape recorder via the recorder's remote jack.

The record, playback, rewind, and fast-forward levers must still be depressed by hand, of course. Nevertheless, once the read and write operations are started, the AIM 65 will take full control over both recorders. The AIM 65 will start and stop each recorder at the proper time. (Incidentally, to see your AIM 65 BASIC program doing this will really impress you.)

AIM 65 Software

As we mentioned in our first article, (26:61) AIM 65 tape I/O is buffered. That is, when a character is to be read from, or written to, a tape file, the character is actually read from, or written to, a dedicated area of RAM. This dedicated RAM is known as a buffer. Only when all the data has been read from a buffer, or when the buffer has been filled with data, does the AIM 65 start up a tape recorder to do the actual read or write.

There is a buffer pointer associated with a buffer. The buffer pointer is used by the AIM 65 to keep track of the data within the buffer. Normally, the AIM 65 uses a single buffer which is located in page 1 at $0116. This is the source of our restriction which stated that an input file and an output file could not be open at the same time.

Let's see what would happen if we violated this restriction. Suppose our program were doing a read, and then a write. The very first read would fill the tape buffer with data and set the buffer pointer. The write operation would then store data in the buffer. This would destroy some or all of the input data. That is not all. The write would also change the contents of the buffer pointer. So, when the program did the next read, there would be strange data in the buffer and a pointer pointing to who-knows-where. The result: chaos.

The designers of the AIM 65 anticipated these problems. They provided for a second tape buffer which is located in page 0 at $00AD. This second buffer is activated automatically if the AIM 65 detects that the active input device (INFLG) and the active output device (OUTFLG) are both tape recorders.

Unfortunately, this feature of the AIM 65 will not work for us with BASIC. First of all, the second tape buffer, being in page zero, will obliterate BASIC's page zero variables. This would also cause chaos. Secondly, our text file input and output subroutines set INFLG and OUTFLG for tape operations only long enough to read or write a record. Thus, both INFLG and OUTFLG will probably not be set for tape at the same time. In the sections that follow, we will present one solution to the tape buffer problem. (You may come up with other solutions.)

Synchronization Bytes

Each AIM 65 tape data block is preceded by a series of synchronization bytes. The contents of a RAM variable called GAP determine the number of synchronization bytes written. GAP is initialized by the AIM 65 to a value of $08. The AIM 65 manuals recommend, however, that GAP be manually set to $80 when the tape recorder is going to be used in a start-stop mode such as when loading a BASIC program. Experiment with the setting for GAP. A smaller value will result in faster read and write times. Don't set GAP too small, however, or the tape will become unreadable. We have found that a value of $20 works perfectly for three different tape recorders.

If you are willing to do a little surgery on your tape recorder, you may be able to achieve even faster performance. We modified a Radio Shack CTR-40B so that its electronics would remain on, even when the motor was turned off. We now get good start-stop results with the default value of GAP—$08. By the way, this modification changed the motor type from a type IV to a type III.

Using Dual Cassettes

With the proper software, the AIM 65 is fully capable of dual cassette operations with BASIC. We will use our text file input and output subroutines, plus additional software which is described later.

Of course, the first step to take is to connect two tape recorders to your AIM 65. Dedicate one of them as an input drive and the other as an output drive. (It doesn't matter which is which, but we use drive 1 for input and drive 2 for output.) Test each recorder thoroughly. Make absolutely sure that the motor control circuits are working. Experiment a bit and find the best value of GAP for your system. You should now be assured of obtaining reliable tape operations.

Standard Motor Control

When we designed the text file input and output subroutines, we made some decisions as to when to turn a given tape recorder on or off. In addition, the AIM 65 itself toggles the recorders at certain times. We have summarized the times when the tape recorders are turned on and off.

Read Operation

Reading a block:	only the input tape recorder is turned on
Between blocks:	both tape recorders are turned off
End of file:	only the input tape recorder is turned on

Write Operation

Writing a block:	only the output tape recorder is turned on
Between blocks:	both tape recorders are turned off
End of file:	both tape recorders are turned on

In most cases, this is a fairly convenient way to control the tape recorders. For example, reads and writes can easily be alternated.

For ease of operation, there are two guidelines to follow:

1. Position *both* the input and output tapes *before* you begin processing.

2. Do not close an output file until you have finished all input and output processing.

Let's examine these guidelines. Suppose we are going to read a tape and then write some of the data to an output tape. As soon as we read the very first block of data from the input tape recorder, the output tape recorder will be turned off. If we haven't already positioned the tape, we would have to remove the plug from the recorder's remote jack to do so.

Figure 1

```
7D41 AD 3F 7D          LDA   INBFPT   MOVE INPUT BUF PNTR
7D44 8D 36 A4          STA   TAPTR    TO AIM BUF PNTR
7D47 A2 4F             LDX   #$4F
7D49 BD 8D 7D   LBLA   LDA   INBUFR,X MOVE 80 BYTES
7D4C 9D 16 01          STA   FORMA,X  TO AIM BUFFER
7D4F CA                DEX
7D50 10 F7             BPL   LBLA
7D52 20 00 7C          JSR   TEXTIN   CALL TEXT INPUT SUBR
7D55 AD 36 A4          LDA   TAPTR    MOVE AIM BUF PNTR
7D58 8D 3F 7D          STA   INBFPT   TO INPUT BUF PNTR
7D5B A2 4F             LDX   #$4F
7D5D BD 16 01   LBLB   LDA   FORMA,X  MOVE 80 BYTE AIM BUF
7D60 9D 8D 7D          STA   INBUFR,X TO INPUT BUF
7D63 CA                DEX
7D64 10 F7             BPL   LBLB
7D66 60                RTS
7D67 AD 40 7D          LDA   OUBFPT    OUTPUT BUFF. PTR.
7D6A 8D 36 A4          STA   TAPTR     TO AIM BUFF. PTR.
7D6D A2 4F             LDX   #$4F
7D6F BD DD 7D   LBLC   LDA   OUTBFR,X 80 BYTE OUTP. BUFF.
7D72 9D 16 01          STA   FORMA,X  TO AIM BUFFER
7D75 CA                DEX
7D76 10 F7             BPL   LBLC
7D78 20 A4 7C          JSR   TXTOUT    CALL TEXT OUTP. SUBR.
7D7B AD 36 A4          LDA   TAPTR     MOVE AIM BUFF. PTR.
7D7E 8D 40 7D          STA   OUBFPT   TO OUTP. BUFF. PTR.

7D81 A2 4F             LDX   #$4F
7D83 BD 16 01   LBLD   LDA   FORMA,X MOVE 80 BYTE AIM BUFF.
7D86 9D DD 7D          STA   OUTBFR,X TO OUTPUT BUFFER
7D89 CA                DEX
7D8A 10 F7             BPL   LBLD
7D8C 60                RTS
```

In the second case, suppose that we had wanted to read some more data from the input tape, even though we had finished writing the output tape. As soon as we close the output file, both tape recorders would be turned on (the monitor routine DU11 does this). By the time we get around to reading the next block, we would have found that the tape had already started. There is a good chance then of misreading one or more blocks.

Controlling the Motors with BASIC

You can also control the tape recorders with BASIC. Bits 4 and 5 of AIM 65 port B (at $A800 or 43008) are used to toggle the motor control circuits. Bit 4 controls drive 1, and bit 5 controls drive 2. A one in the proper bit will turn a motor on, while a zero will turn the motor off.

With the use of PEEK and POKE statements, BASIC can access port B and set the appropriate bits. The chart below shows the code to do this.

To turn a motor on:

POKE 43008,(PEEK(43008) or K)

Action	K
Turn drive 1 on	16
Turn drive 2 on	32
Turn both drives on	48

To turn a motor off:

POKE 43008,(PEEK(43008) and K)

Action	K
Turn drive 1 off	239
Turn drive 2 off	223
Turn both drives off	207

Naturally, you should be careful about controlling the motors this way. If a motor is turned on while you are in the middle of processing a file, you can imagine the kind of errors that could result. A good rule of thumb is to use the standard motor control options whenever possible. Resort to BASIC only when you need to—to turn both drives on after an I/O error for example.

Compatibility with Input and Output Subroutines

Our AIM 65 text file input and output subroutines were designed to be incorporated into a dual cassette file

Figure 2

```
10  REM RENUMBER PROGRAM
20 REM OPEN INPUT FILE
30 POKE 245,0: REM $F5
40 REM OPEN OUTPUT FILE
50 POKE 247,0:REM $F7
60 INPUT "STARTING LINE";SL
70 INPUT"INCREMENT";INC
80 REM INITIALIZE LINE NUMBER
90 LN = SL
100 REM READ A LINE OF BASIC TEXT
110 A$=""
120 FOR I=1 TO 80
130 A$=A$+"*"
140 NEXT I
150 POKE 4,65:REM $41
160 POKE 5,125:REM $7D
170 L=USR(0)
180 REM ERROR TEST
190 IF L<0 THEN STOP
200 REM END FILE TEST
210 IF L=0 THEN 370
220 REM GET LENGTH OF ORIGINAL LINE #
230 A$=LEFT$(A$,,L)
240 LL=LEN(STR$(VAL(A$)))
250 REM PUT IN THE NEW LINE #
260 A$=STR$(LN)+MID$(A$,LL+1)
270 LN=LN+INC
280 REM OUTPUT THE LINE
290 POKE 4,103:REM $67
300 POKE 5,125:REM $7D
310 Z=USR(LEN(A$))
320 REM ERROR TEST
330 IF Z<0 THEN STOP
340 REM DO THE NEXT LINE
350 GOTO 100
360 REM CLOSE THE OUTPUT FILE
370 A$=CHR$(26):REM CONTROL Z
380 POKE 4,103:REM $67
390 POKE 5,125:REM $7D
400 Z=USR(1):REM OUTPUT CONTROL Z
410 Z=USR(0):REM CLOSE THE FILE
420 PRINT"***DONE***"
430 END
```

Figure 3

```
10 REM DEMO PROGRAM FOR BASIC RENUMBER
15 FOR I=1 TO 10
16 PRINT I;
17 PRINT SQR(I)
20 NEXT I
25 REM GOTOS AND GOSUBS ARE NOT RENUMBERED
30 GOSUB 100
35 GOTO 15
100 REM A SUBROUTINE
105 LET C=C+1
110 PRINT C
115 RETURN
```

This is the program after renumbering.

```
120 REM DEMO PROGRAM FOR BASIC RENUMBER
160 PRINT I;
180 PRINT SQR(I)
200 NEXT I
220 REM GOTOS AND GOSUBS ARE NOT RENUMBERED
240 GOSUB 100
260 GOTO 15
280 REM A SUBROUTINE
300 LET C=C+1
320 PRINT C
340 RETURN
```

system. They both use the variable A$ for holding a record. When a record is read, the input subroutine tells you how many bytes of data were actually stored in A$. When you write a record, you tell the output subroutine how many bytes from A$ to write. Now, to read a record and to write it back to tape, all we have to do in BASIC is:

```
POKE 4,low address of input
     subroutine
POKE 5,high address of input
     subroutine
L=USR(0): REM READ A
     RECORD INTO A$
POKE 4,low address of output
     subroutine
POKE 5,high address of output
     subroutine
Z=USR(L): REM WRITE A
     RECORD FROM A$
```

Thus, dual cassette input and output becomes very easy.

Buffer Management

Before we can begin actually using the dual cassette interface with BASIC, we need to overcome the AIM 65's single buffer problem. The approach we are taking is a brute force method. It does have the virtue, however, of being very simple. We call this simple software a buffer manager.

We will proceed by setting up our own input buffer and pointer, and our own output buffer and pointer. Each buffer will be 80 bytes long—the same as the AIM 65's. Whenever we request an input or output operation, the buffer manager will do the following:

1. Move our buffer and pointer to the AIM 65's buffer and pointer
2. Perform the read or write operation
3. Move the AIM 65's buffer and pointer back to our buffer and pointer.

Although there is a lot of data flying back and forth in RAM, the buffer manager keeps everything straight. It allows us to have an input and an output file open at the same time.

Loading the Buffer Manager

The code for the buffer manager is shown in AIM 65 instruction format in figure 1. A hex dump is not included, due to the degree of customization that will be required for your system.

There are two routines shown in figure 1. One handles text file input and the other text file output. These routines will now become the main entry points for all I/O operations. This means that whenever you want to read or write text files, your BASIC program should call the appropriate buffer manager and *not* the input or output subroutines themselves.

To load the buffer manager in your system, you need to do the following:

1. Set aside four areas of RAM for a total of 162 bytes.

Area	**Length**
Input buffer pointer	1
Ouput buffer pointer	1
Input buffer area	80
Output buffer area	80

2. Modify the code shown in figure 1 to use the pointer and buffer addresses which you have just established.
3. Make sure that the buffer manager's references to the text file input and output subroutines are correct.

Once you are satisfied that everything is correct, be sure to save the programs on tape.

The total memory requirements to support full dual cassette operations now become:

Text file input subroutine	164
Text file output subroutine	148
Buffer manager code	76
Buffer manager RAM	162
TOTAL	550

Use this figure when responding to BASIC's MEMORY SIZE prompt. If you have a 4K AIM 65, you would respond with 3546 (4096 minus 550).

Do not let the 550 byte memory requirement worry you. There is still room left for a pretty good size BASIC program. Also, since you now have dual cassette capabilities, you can work on files that are larger than the available RAM in your system!

Sample Program

Our sample program is a very simple BASIC renumber program. It only renumbers the lines. It will not update GOTOs and GOSUBs. They will have to be corrected manually.

The listing for the renumber program is shown in figure 2. We have included a sample run in figure 3. If you've got everything working properly, you should be able to get exactly the same results that we did. To use the renumber program, follow this step-by-step procedure.

1. Save the program you wish to renumber on tape.
2. Make sure that you've loaded the machine language programs and that BASIC is initialized accordingly.
3. Key in or LOAD the renumber program.
4. Put the tape containing the program to be renumbered in the input tape recorder.
5. Place a blank tape in the output tape recorder.
6. Space the blank tape to a point past the leader.
7. Type RUN to begin the renumbering.
8. You will be asked for a new starting line number and increment. Respond with the desired numbers.
9. You will be asked for the input device, file name, and tape drive number. Respond appropriately.
10. You will be asked for the output device, file name, and tape drive number. Respond appropriately.
11. Sit back and watch the blocks of data being read, processed, and written.
12. Rewind both tapes when the renumber program is done.
13. Load the renumbered program.
14. Correct any GOTOs and GOSUBs.

The operation of the renumber program is very simple. It works by reading one line of the original program at a time. It then strips off the line number and replaces it with a new line number. Finally, it writes out the modified line to the output tape recorder.

You may want to expand this simple renumber program into a full renumber program that fixes GOTOs and GOSUBs automatically. To do this, you will probably need to read the original program twice. The first time around, renumber the lines as we have already done. At the same time, build an array of old line numbers and the corresponding new line numbers. The second time that you read the tape, test each line for a GOTO or GOSUB. If one is present, you can find out what the new destination should be by referring to the table that you built the first time around. (This, of course, is not the best way to renumber BASIC programs. It is, however, a pretty good way to become familiar with file operations in BASIC.)

The program in figure 2 is straighforward. We followed all the procedures described in earlier articles for text file input and output. The only difference is that our POKEs to locations 4 and 5 set up references to the buffer managers and *not* to the text file I/O routines. One thing to remember is that you must always write a control-z to the end of any text file that you plan on LOADing with BASIC.

Conclusion

This series of articles has described a way to make your AIM 65 a powerful data processor. We began by describing a way to read AIM 65 text files with BASIC. Next we added the capability to write text files. Finally, we combined these components into an integrated dual cassette system.

With the expenditure of 550 bytes of RAM and two tape recorders, your AIM 65 has almost all the capabilities of a floppy disk-equipped system. While we do not have the ability to do random access, we have practically unlimited storage capabilities. AIM 65 cassette I/O is fairly fast and the dual cassette interface really works!

There are many applications that are now opened up. For example, we have set up a home accounting system that compares expenditures against a budget. Let us hear about your applications.

Christopher Flynn has an AIM with 32K of RAM and a Model 33 teletype for hardcopy output. His software interests include Assembly language and BASIC. To support his hobby, Chris is employed by the Fairfax County government as a systems analyst for the county's tax systems.

MICRO

Atari Real Time

This is a program which allows drawing in graphics mode by moving the cursor, with the keyboard, in any direction desired. This program also permits the usage of several different colors.

Charley and Mary Kozarski
1035 Fuller NE
Grand Rapids, Michigan 49503

Here is a program which allows drawing in graphics modes 3 through 8, using the keyboard alone. This is designed for Atari 800 or 400 owners who primarily use their computers for other than games. It's great for those who don't have joysticks but would like to doodle in graphics mode occasionally.

Controlling real-time graphics with the keyboard can be accomplished by using an IF...THEN statement in association with William Colsher's famous PEEK (764) command. Lines 45-85 look for a certain key to be pressed and *if* it is, *then* the graphics plot is moved one line in that particular direction.

Line 10 sets the maximum plot dimensions for that graphics mode. The GOSUB routine is used to keep the plot on the screen. Without it, moving the plot beyond the maximum limit for that mode would cause an ERROR 141 (cursor out of range). Also, at this time, the computer would kick out of graphics mode and you'd have to start your drawing from the beginning!

It is possible to change the color of the plot at any time without changing what has already been plotted. This can be done by lines 35 and 45. Also, line 25 allows the present plot position to be indicated by blinking the cursor on and off.

One last thing worth mentioning—one of the plot colors is black or background color, which allows you to erase anything you have already drawn.

We Atari owners would appreciate it if the rest of you Atari fans out there would start sending in programs and information that you think would be interesting or fun.

Charles Kozarski works as an Electronics Technician for a large retailer. Computers have always fascinated him and he decided to explore them as a hobby. He owns an Atari 800 and spends a great deal of time developing programs.

Mary Kozarski is an underwriter for an insurance agency.

```
  5 GRAPHICS 5+16:X=40:Y=24:C=1
 10 XMAX=80:YMAX=48:REM-PLOT LIMITS
 15 POKE 764,255:REM-CLEAR LAST KEY
 20 REM(25)-BLINKS PRESENT PLOT POINT
 25 COLOR 0:IF C=4 THEN COLOR 2
 30 PLOT X,Y:FOR D=1 TO 50:NEXT D
 35 COLOR C: PLOT X,Y: REM-SET COLOR
 40 REM(45)-SET PLOT DIRECTIONS TO KEYS
 45 IF PEEK (764)=18 THEN C=C+1: IF C>4 THEN C=1
 50 IF PEEK (764)=6 THEN X=X-1: GOSUB 95
 55 IF PEEK (764)=7 THEN X=X+1: GOSUB 95
 60 IF PEEK (764)=14 THEN Y=Y-1: GOSUB 95
 65 IF PEEK (764)=15 THEN Y=Y+1: GOSUB 95
 70 IF PEEK (764)=48 THEN X=X-1:Y=Y-1: GOSUB 95
 75 IF PEEK (764)=50 THEN X=X+1:Y=Y-1: GOSUB 95
 80 IF PEEK (764)=54 THEN X=X-1:Y=Y+1: GOSUB 95
 85 IF PEEK (764)=55 THEN X=X+1:Y=Y+1: GOSUB 95
 90 GOTO 15
 95 REM=SET PLOT LIMITS!
100 IF X>=XMAX THEN X=0
105 IF Y>=YMAX THEN Y=0
110 IF X<0 THEN X=XMAX-1
115 IF Y<0 THEN Y=YMAX-1
120 RETURN
```

KEY

Key		Action
↑ / –	=	Move Plot Up
↓ / =	=	Move Plot Down
← / +	=	Move Plot Left
→ / *	=	Move Plot Right
C	=	Change Plot Color
(/ 9	=	Move Plot Up Diagonal Left
) / 0	=	Move Plot Up Diagonal Right
CLEAR / <	=	Move Plot Down Diagonal Left
INSERT / >	=	Move Plot Down Diagonal Right

MICRO

New Publications

Mike Rowe
New Publications
P.O. Box 6502
Chelmsford, MA 01824

This column lists new publications received for review and also reports on pertinent publication announcements received from book and periodical publishers. Some works mentioned here may be reviewed by MICRO at a later date.

Ohio Scientific

Introductory works from Elcomp Publishing, Inc. (3873L Schaefer Avenue, Chino, California 91710), both by J. Clothier and W. Adams, have confusing titles and have been promoted in a confusing way: **The First Book of Ohio Scientific Vol. I** (publisher's order no. 157) and **The Second Book of Ohio Scientific** (order no. 158). The first book carries an announcement for the second book (order no. 158) under the title *The First Book of Ohio Scientific, Vol. II.* Both books are largely compilations of material issued in various formats by the manufacturer or previously published elsewhere. Here are brief descriptions:

The first book (paperbound, 188 pages, $7.95) covers the Challenger 1P and 1P MF, the Superboard II, the Challenger 4P and 4P MF, the Challenger 8P and 8P DF, peripherals, software, and other introductory information.

The second book (paperbound, 188 pages, $7.95) covers the OS-65 (version 2.0 disk operating system, microcomputer operating system, string variables, machine code, and the editor); the wordprocessor WP-2; I/O drivers; memory test program; and various other hardware and software data.

PET

PRINTOUT, a magazine for users of PET and Commodore systems, published ten times a year in the United Kingdom, is now offered to readers in the U.S. It covers hardware, reviews software, contains articles on programming and applications, and publishes photos and listings. A sample issue is available postpaid for US$3.00 and a subscription for US$36.00 from *PRINTOUT*, P.O. Box 48, Newbury, Berkshire RG16 OBD, England.

General 6502

6502 Applications Book by Rodnay Zaks. 6502 Series, Volume III, Sybex (2344 Sixth Street, Berkeley, California 94710), 1979, 278 pages plus advertisements, over 200 illustrations, 5½ × 8½, paperbound.
ISBN: 0-89588-015-6 $12.95

Covers application techniques for the 6502. The book assumes an elementary knowledge of microprocessor programming on the level of the series' preceding volume, *Programming the 6502.*

CONTENTS: *Introduction. The Input Output Chips*—Introduction; Basic Definitions; The 6502 PIA; The 6522; Programming the 6522; The 6530 ROM-RAM I/O Timer (RRIOT); The 6532; Summary. *6502 Systems*—Introduction; Standard 6502 System; The KIM-1; The SYM-1; The AIM 65; Other boards. *Basic Techniques*— Introduction. Section I, The Techniques: Relays; Switches; Speaker; A Morse Generator; Time of Day Clock; A Home Control Program; A Telephone Dialer. Section II, Combinations of Techniques: Introduction; Generating a Siren Sound; Sensing an Input Pulse; Pulse Measurement; A Simple Music Program; KIM Traffic Control; Learn the Multiplication Table; Summary. *Industrial and Home Applications*—Introduction; A Traffic Control System; Dot Matrix LED; Displaying Switch Values; Tone Generation; Music; A Burglar Alarm; DC Motor Control; Analog to Digital Conversion (A Heat Sensor); Summary. *The Peripherals*—Introduction; Keyboard; Paper Tape Reader or ASCII Keyboard; Microprinter; Summary. *Conclusions. Appendix A: A 6502 Assembler in BASIC*—Introduction; General Description; Using the Assembler; Syntax; HP2000FBASIC. *Appendix B: Multiplication Game: The Program. Appendix C: Program Listings. Appendix D: Hexadecimal Conversion Table. Appendix E: ASCII Conversion Table. Appendix F: 6502 Instructions.*

General Microcomputer

Small Business Programs by S. Roberts. Elcomp Publishing, Inc. (3873L Schaefer Avenue, Chino, California 91710), 1980, v, 118 pages, 5½ × 8½, paperbound.
ISBN: 3-921682-57-6 $14.90

Contains 32 brief business programs (averaging about 3 pages each) in Microsoft BASIC. According to the author (who developed them on PET 2001, CBM 3016, the Ohio Scientific Superboard, TRS-80, and the Sharp Computer MZ80K), the programs will run on any BASIC computer.

Microcomputers and Physiological Simulation by James E. Randall. Addison-Wesley Publishing Company, Inc.—Advanced Book Program (Reading, Massachusetts), 1980, xvi, 234 pages, paperbound.
ISBN 0-201-06128-7 $14.50

An introduction to microcomputers and their use in mathematical simulations of physiological processes.

CONTENTS: Foreword by Arthur C. Guyton. Preface. *Introduction*—Mechanical Models; Mathematical Models; Analog Computers; Digital Computers; Teaching by Simulation; References. *Microcomputer Components*—Microprocessors; Semiconductor Memory; Keyboard; Cathode Ray Tube Displays; Mass Storage; Microcomputers; References; Microcomputer Periodicals; Manufacturers Cited. *Operating Systems and Programming Languages*—Monitors and Operating Systems; CP/M; Assemblers; BASIC; Other Programming Languages; References; Software Sources. *Hardware Enhancements for Simulation*—Graphics; Numerical Processors; References; Manufacturers Cited. *Representative Microcomputers*—TRS-80; Apple II; S-100 Bus Microcomputer; References. *Compartmental Kinetics: A First Example*—The Hydraulic Model; Computed Responses; The BASIC Program; References; Chapter Appendix. *The Glucose Tolerance Test*—The Insulin-Glucose Interaction Model; The BASIC Program; Computed Responses; References; Chapter Appendix. *Cardiovascular System Mechanics*—The Functional Relationships; Steady-State Solutions; Steady-State Exercises; Transient Solutions; References; Chapter Appendix. *Arterial Pulse Pressure*—The Model; Computed Responses; The BASIC Program; References; Chapter Appendix. *Vectorcardiography and the Limb Leads*—Computed responses; The BASIC Program; Chapter Appendix. *Distortion of Waveforms*—Computed Responses; Digital Filtering; Restoring Distorted Waveforms; The BASIC Program; References; Chapter Appendix. *Axon Action Potentials*—Formulation in BASIC; Output Displays; Properties of Excitation; Computation Methods; References; Chapter Appendix. *Cardiac Action Potentials*—Formulation in BASIC; Computed Responses; Output Display and Computation Methods; References; Chapter Appendix. *Formatting Student Exercises*—Turnkey Systems; Programming. *Index.*

Full Disassembly Listing on Small Systems

This is a utility program for the small system owner who publishes software listings. It examines a program in memory and lists the program in disassembly format, identifying the operands of each op code. After printing the op code and the associated operands, it pauses to allow the operator to furnish label names and comments. Thus, a "camera-ready" listing can be produced even though the system doesn't have a large memory.

Ralph Tenny
P.O. Box 545
Richardson, Texas 75080

The First Book of KIM contained MINIDIS, by Dan Lewart. This program allowed the user to scan memory and identify any illegal 6502 op codes by blinking the KIM display. After I converted MINIDIS to drive a printer, I discovered that it allowed me to print each op code and the corresponding operands. However, it would also "disassemble" many other bytes and identify each according to the type of instruction it "ought" to be. After considerable thought, I began over and produced DISEDIT II, which would test for possible 6502 illegal op codes and sort the legal op codes according to the number of operands each uses. After each op code/operand combination, DISEDIT II would halt to allow the user to type labels, mnemonics and comments in a form to gladden the heart of any editor. This program functioned on an unexpanded KIM, but did not leave room for a large program.

DISEDIT II didn't really disassemble the code. If it had (typing out addresses, code and operands), it would also have typed in locations instead of labels, and would not (unless modified) have allowed for an opportunity to type in comments.

In the process of expanding my KIM, it got to be a hassle having both a program area and a zero-page look-up table, so the program was modified to be fully relocatable and ROMable. I will eventually have it in a ROM plug-in module. The version shown is located at 1200_{16} and uses the following zero-page buffers:

- 0000_{16} — TEMP; temporary storage of op code being tested.
- 0002_{16} — SAL; low byte of starting address of program being listed.
- 0003_{16} — SAH; high byte of start address.
- 0004_{16} — EAL; low byte of end address in auto list mode.
- 0005_{16} — EAH; high byte of end address.
- 0007_{16} — TMPY; storage for current *Y*-index value.

Auto list mode is a last-minute addition which allows the listing to proceed from program start to end, without pausing for you to enter the labels, etc. This speeds up the debugging utility of the program considerably, and allows a quick check on accuaracy of program entry when keying in a new program. To activate the auto list mode, simply change two locations as follows: $12DD_{16}$ — 08; $12E3_{16}$ — 02. This modification forces the branch to pass up STOP1, and the program runs continuously. If you have implemented sense switches, as I plan to, the program could be modified slightly to test a sense switch and run in auto list or normal mode, at the setting of one switch.

After having the program around for a while, other uses for it have become apparent. For example, it requires quite a bit of concentration to check keyed-in data against the usual assembly-format listing. However, using DISEDIT IV presents the memory contents in the same format as the published listing. This speeds up entry checking and recognition of errors. Also, an illegal op code which results from an improper key entry will break up the entire pattern, so only operands require close checking to verify their accuracy.

Here is how to use the program: The user places the address of the first byte of executable code in 0002_{16} and 0003_{16} with the low byte first as usual with KIM. Note the comment after address $12E2_{16}$. If the code is to be modified for an exit to monitor, etc., then enter the end address in 0004_{16} and 0005_{16}

After loading the start address, simply go to location 1200_{16} and hit GO. The printer will print the first line of machine code and wait for you to space over to the appropriate columns and type in labels, mnemonics, operands and comments for that line. Hit GO again, type, etc., and do this until you hit the end of the program. Of course, if you're only debugging, print out the whole program as fast as you can hit GO. (You could also change to auto list mode as outlined above.)

Meanwhile, the program works this way: The starting address is printed out, by the code beginning at label START. Next, the *Y* index is loaded (0, on the first pass) and used to fetch the first byte of code. At location $121B_{16}$ the lower nibble of the op code is stripped off and loaded into the *X* register.

```
1200                TEMP   *      $0000
1200                SAL    *      $0002
1200                SAH    *      $0003
1200                EAL    *      $0004
1200                EAH    *      $0005
1200                TMPY   *      $0007
1200                VCTR   *      $1780
1200                CRLF   *      $1E2F
1200                PRTBYT *      $1E3B
1200                OUTSP  *      $1E9E
1200                OUTCH  *      $1EA0

1200 A9 00          INIT   LDA    #$00      CLEAR ACCUMULATOR
1202 AA                    TAX              CLEAR X INDEX
1203 A8                    TAY              AND Y INDEX
1204 85 00                 STA    TEMP      CLEAR TEMP. STORAGE
1206 85 07                 STA    TMPY      AND INDEX BUFFER
1208 A5 03          START  LDA    SAH       GET START ADDR.--HIGH
120A 20 3B 1E              JSR    PRTBYT    AND PRINT IT.
120D A5 02                 LDA    SAL       SAME WITH
120F 20 2E 12              JSR    PRINT     START LOW.
1212 20 9E 1E              JSR    OUTSP     PRINT A SPACE
1215 A4 07                 LDY    TMPY      GET CURR. INDEX VAL.
1217 B1 02                 LDA    (SAL),Y   & GET OP CODE.
1219 85 00                 STA    TEMP      REMEMBER THE OP CODE
121B 29 0F                 AND    #$0F      MASK OFF LOW NIBBLE.
121D AA                    TAX              AND USE AS INDEX
121E BD EC 12              LDA    TABL,X    TO GET A JUMP VECTOR.
1221 8D 80 17              STA    VCTR      SET UP IND. VECT. LOW
1224 A9 12                 LDA    #$12      GET PAGE OF JUMP DEST.
1226 8D 81 17              STA    VCTR      +01 AND COMPLETE VECTOR.
1229 A5 00                 LDA    TEMP      GET CURRENT OP CODE
122B 6C 80 17              JMP    (VCTR)    AND PROCESS IT.
122E 20 3B 1E       PRINT  JSR    PRTBYT    PRINT ACCUM. CONTENTS
1231 20 9E 1E              JSR    OUTSP     FOLLOWED BY A SPACE.
1234 60                    RTS              RETURN
1235 A5 00          QSMK   LDA    TEMP      GET INVALID OP CODE
1237 20 2E 12              JSR    PRINT     AND PRINT IT.
123A A9 3F                 LDA    #$3F      LOAD ASCII FOR '?'
123C 20 A0 1E              JSR    OUTCH     AND PRINT IT.
123F 20 2F 1E              JSR    CRLF      NEW LINE
1242 A4 07                 LDY    TMPY      GET CURRENT INDEX,
1244 C8                    INY              BUMP IT TO NEXT BYTE
1245 4C C5 12              JMP    NEXT      NEW OP CODE LOCATION
1248 C9 20          TSTA   CMP    #$20      IS THIS 'JSR'?
124A F0 60                 BEQ    THREE     IF SO, PRINT 2 OPERANDS.
124C C9 80                 CMP    #$80      INVALID OP CODE?
124E F0 E5                 BEQ    QSMK      IF SO, QUESTION IT
1250 C9 40                 CMP    #$40      HOW ABOUT 'RTI'?
1252 F0 4E                 BEQ    ONE       PRINT ONLY OP CODE.
1254 C9 60                 CMP    #$60      RTS?
1256 F0 4A                 BEQ    ONE       SAME DEAL!
1258 C9 00                 CMP    #$00      IS IT 'BRK'?
125A F0 46                 BEQ    ONE       IF SO PRINT ONE BYTE
125C D0 49                 BNE    TWO       REST ARE 2-BYTE OP CODES
125E C9 A2          TSTB   CMP    #$A2      HAVE WE 'LDX'?
1260 F0 45                 BEQ    TWO       THEN PRINT TWO BYTES
1262 D0 D1                 BNE    QSMK      OTHERWISE, INVALID
1264 C9 24          TSTC   CMP    #$24      IS THI 'BIT'?
1266 F0 3F                 BEQ    TWO       THEN PRINT TWO
1268 C9 E4                 CMP    #$E4      IT MIGHT BE CPX
126A F0 3B                 BEQ    TWO       SURE ENOUGH, PRINT TWO
126C C9 84                 CMP    #$84      TRY HEX 84
126E 90 C5                 BCC    QSMK      THROW OUT THOSE LESS
```

```
1270 C9 C4             CMP   #$C4     IS IT CPY?
1272 B0 C1             BCS   QSMK     THROW OUT THOSE GRTR.
1274 90 31             BCC   TWO      REST ARE 2-BYTE CODES
1276 C9 89     TSTD    CMP   #$89     GOT AN ILLEGAL?
1278 F0 BB             BEQ   QSMK     YEP, THROW IT OUT
127A 29 10             AND   #$10     SORT ODDS FROM EVENS
127C F0 29             BEQ   TWO      TWO BYTE CODES
127E D0 2C             BNE   THREE    & REST ARE 3-BYTE CODES.
1280 C9 9A     TSTE    CMP   #$9A     FIND AN ODD ONE
1282 F0 1E             BEQ   ONE      AND LOG IT.
1284 C9 BA             CMP   #$BA     GET AN EVEN ONE,
1286 F0 1A             BEQ   ONE      COUNT IT AS A ONE,
1288 29 10             AND   #$10     SORT OUT THE EVEN ONES,
128A D0 A9             BNE   QSMK     AND REJECT THE ODD ONES.
128C F0 14             BEQ   ONE      CALL REST ONE-BYTE CODES.
128E C9 0C     TSTF    CMP   #$0C     THIS ONE IS UNUSED
1290 F0 A3             BEQ   QSMK     SO REJECT IT.
1292 C9 BC             CMP   #$BC     THIS ONE USED--TRY IT
1294 F0 16             BEQ   THREE    THREE BYTE CODE
1296 29 10             AND   #$10     SORT OUT ODD ONES
1298 F0 12             BEQ   THREE    KEEP THE EVEN ONES
129A D0 99             BNE   QSMK     BRAND THE ODD ONES
129C C9 9E     TSTG    CMP   #$9E     ONLY ONE MORE TO TRY
129E F0 95             BEQ   QSMK     GOT HIM
12A0 D0 0A             BNE   THREE    KEEP REST AS THREE'S
12A2 A5 00     ONE     LDA   TEMP     GET CURRENT OP CODE
12A4 4C C2 12          JMP   PRTONE   AND PRINT IT ALONE
12A7 A5 00     TWO     LDA   TEMP     GET OP CODE
12A9 4C B8 12          JMP   PRTTWO   AND GIVE IT A FRIEND
12AC A5 00     THREE   LDA   TEMP     GET OPCODE
12AE 20 2E 12  PRTTHR  JSR   PRINT    AND GIVE IT A FAMILY!
12B1 A4 07             LDY   TMPY     GET CURRENT INDEY
12B3 C8                INY            &BUMP IT ONE.
12B4 B1 02             LDA   (SAL),Y  USE TO FETCH NEXT BYTE
12B6 84 07             STY   TMPY     KEEP THE INDEX VALUE
12B8 20 2E 12  PRTTWO  JSR   PRINT    PRINT THE BYTE
12BB A4 07             LDY   TMPY     GET THE INDEX
12BD C8                INY            ADD ONE
12BE B1 02             LDA   (SAL),Y  AND GET ANOTHER BYTE.
12C0 84 07             STY   TMPY     REMEMBER THE INDEX
12C2 20 2E 12  PRTONE  JSR   PRINT    PRINT A LONER
12C5 E6 07     NEXT    INC   TMPY     SET UP FOR NEXT PASS
12C7 A5 07             LDA   TMPY     BY ADDING THE INDEX.
12C9 A0 00             LDY   #$00     CLEAR Y INDEX
12CB 84 07             STY   TMPY     AND TMPY.
12CD 18                CLC            PREPARE TO ADD
12CE 65 02             ADC   SAL      THE ACCUMULATOR TO SAL.
12D0 85 02             STA   SAL      UPDATE SAL
12D2 A5 03             LDA   SAH      AND GET HI BYTE
12D4 69 00             ADC   #$00     BUMP IT IF CARRY
12D6 85 03             STA   SAH      AND UPDATE SAH.
12D8 A5 02             LDA   SAL      GET CURRENT POINTER
12DA C5 04             CMP   EAL      = LAST LOCATION?
12DC 90 06             BCC   STOPA    WAIT FOR SLOW TYPIST!
12DE A5 03             LDA   SAH      TEST SAH
12E0 C5 05             CMP   EAH      LAST BYTE?
12E2 90 00             BCC   STOPA    OPTIONAL PROV. FOR MON.
                                      EXIT
12E4 00        STOPA   BRK            WAIT FOR SLOWPOKE
12E5 00                BRK            FILLER BYTE
12E6 20 2F 1E          JSR   CRLF     BRK STOPS IT HERE--
                 PRESS GO
12E9 4C 08 12          JMP   START    GO BACK FOR MORE
12EC  48 A7 5E 37 64 A7 A7 37 A2 76 80 37 8E AC 9C 37
```

This index is used to locate a jump vector from the table at the end of the program. Note that this vector is stored at 1780_{16} and that 12_{16} is loaded into 1781. At $122B_{16}$ an indirect jump takes you to the particular sorting routine which tests the op code to see if it is legal. If it is, the sort routine also sets up to print the operands for that op code.

For example, let's assume that 20_{16} is the op code under consideration. After the mask step at $121B_{16}$ $X = 0$. The jump vector at $12FA_{16}(X)$, where X is 0, is 48_{16} . In other words, the program goes to 1248_{16} to see if it has a legal op code.

At 1248_{16} the first test is for 20_{16} which tests true. The program goes directly to label THREE ($12AC_{16}$), where the op code is picked up from TEMP and printed. Since 20_{16} is a three-byte op code, the program continues on through labels PRT2 and PRT1, then on to NEXT, which adds the current Y index (now up to 02 after having printed the JSR op code and the associated two address bytes), and adds it to the start address in SAL. Next, SAH is updated if there was a carry, and TMPY zeroes TMPY so that location 1215_{16} will begin with $Y = 0$ again. The program continues until you stop it, or it stops by finding it has qualified on the last address to be printed out. In the case of a two-byte code, the jump is to TWO and then PRT2; for a single-byte code the jump sequence is ONE, PRT1.

If you wish to locate this program in another area of memory, simply change all page numbers (the third byte in any three-byte codes) from 12 to the desired location. For example, to begin the program at 0200_{16} simply change all 12's in the third byte to 02. Note that all the other three-byte codes refer to KIM monitor locations, which must remain unchanged.

Ralph Tenny has worked in the electronics industry since 1954, and was working at Texas Instruments when micros came of age. In recent years, he has been working for George Goode & Associates, Inc.. Besides general applications of micros in laboratory work there, he teaches Assembly language programming on the Texas Instruments TMS 9900 microprocessor. He has had a KIM-1 for about four years. Part of his work has been writing about micros—helping write a textbook, application notes, and instruction and user manuals.

MICRO

SYM Bridge Trainer

An elementary program designed to operate on a SYM-1, and requiring less than ½K of RAM. It can easily be adapted to the KIM, or any other 6502. Its general principles can be used by anyone.

Len Green
15 Yotam Street
Achuza
34 675 Haifa, Israel

A somewhat elderly physics teacher with no previous knowledge of computers, and shamefully little of electronics, I purchased one of the first SYM-1's in summer 1978. There was then no documentation available for novices like me. Thanks to MICRO, and more recently *SYMphysis*, the situation has improved immensely, and I have acquired much more SYM material during the past few months than in the whole previous 1½ years. MICRO and other national magazines have published articles during the past year on clocks, music-makers, Morse-Coders, typing trainers, etc., some of them overlapping; also games such as Nim, Mastermind, Noughts and Crosses and Lunar Lander. However, to the best of my knowledge, nothing has been published on my obsession prior to microprocessors, that is—bridge. The following is my undoubtedly crude and amateur attempt to rectify this omission.

```
                   LEN GREEN:
                   BRIDGE TRAINER

                   ZERO PAGE ADDRESSES RESERVED:

                   $00B0 TO $00BC--NORTH'S HAND
                   $00BD TO $00C9--EAST'S HAND
                   $00CA TO $00D6--SOUTH'S HAND
                   $00D7 TO $00E3--WEST'S HAND

                   $00E4 TO $00EC--SCRATCHPAD
                   $00F4 TO $00F7--SCRATCHPAD

                   $00ED--NOTRUMP FLAG
                   $00EE--HIGH-CARD POINTS (H)
                   $00EF--TOTAL-CARD POINTS (T)
                   $00F0--# OF CLUBS IN HAND "S"
                   $00F1--# OF D
                   $00F1--# DIAMONDS IN HAND "S"
                   $00F2--# OF HEARTS IN HAND "S"
                   $00F3--# OF SPADES IN HAND "S"
                   $00F8--LONGEST & STRONGEST SUIT


                   SYM CALLS:

03B4               MONITR   *    $8000
03B4               DELAY    *    $835A
03B4               BEEP     *    $8972
03B4               ACCESS   *    $8B86
03B4               DUMPT    *    $8E87     NEW MONITOR (MON1.1)
03B4               VIATI    *    $A004
03B4               RIOTTI   *    $A41E
03B4               DISBUF   *    $A640
03B4               TV       *    $A656

03B4               STRTAD   *    $03B4
03B4               ENDAD    *    $03B8
03B4               SEGMNT   *    $03BC
03B4               SUIT     *    $03C0
03B4               VALUE    *    $03C4
```

(continued)

Bridge is an intricate game involving the interaction of four players at the bidding stage and three players during the actual play. To simulate the game on a computer requires an extremely long program, and even such a program may not succeed in covering all of the game's different aspects. Several commercial packages are advertised, but since I haven't seen any of them, I cannot judge their merits and limitations. Conversely, a short program can only deal with some restricted aspect of the game, and hence is somewhat of a gimmick. This probably explains the paucity of bridge (and chess) routines printed in journals. A complete and entertaining game of blackjack, Mastermind and the like, can be compressed into about ½K bytes of machine code, which is a nice length for a magazine article. But with bridge, the same ½K will hardly be enough for the preliminaries and bidding, let alone the actual play itself!

This program only goes as far as the opening bids, but all the shuffling, dealing, arranging, and also opening bid routines together only occupy about ⅓K of RAM. It was designed to display fully on the 7-segment panel of a minimal 1K SYM (or KIM), and the display, timing and beeping routines add another 130 bytes or so, making a total listing of 463 bytes. As it stands, it can only be used to check, quiz or compare opening bids, although it can serve to improve speed in bidding, gain insight into the different configurations of the four hands, etc. However, many of the fundamental mechanisms for calling and playing bridge are present, including all four hands, point counts, suit lengths, short and long suits, no-trump and pre-emptive distributions, etc. This program should provide a basis of routines which could be developed and expanded into a reasonably complete game of bridge, to run on SYM's 8K extended onboard RAM, or possibly to be adapted for BASIC.

To operate, simply key in GO/200/CR. A deck is shuffled and the 52 cards dealt into the four hands N, E, S and W, each hand being arranged into the generally accepted order, high to low, of spades, hearts, diamonds and clubs. North's hand is then displayed card by card. ".n" on the left reminds you that North's hand is always displayed first, and the cards themselves appear on the extreme right. Record North's hand as it beeps away, and after the 13th card, the display blanks for 10 seconds, giving

```
0200                    ORG   $0200

0200 20 86 8B           JSR   ACCESS   UNWRITE-PROTECT
0203 A0 03              LDY   #$03      SYSTEM RAM
0205 20 5B 03           JSR   INIT     FOR NORTH
0208 20 1B 02           JSR   DEAL      & ROUND NORTH
020B A9 03     PLAY     LDA   #$03     ROUND E,S,& W
020D 48                 PHA
020E 68        LBLA     PLA
020F A8                 TAY
0210 88                 DEY
0211 98                 TYA
0212 48                 PHA
0213 20 7C 03           JSR   INITA    FOR E,S, & W
0216 68                 PLA
0217 48                 PHA
0218 D0 F4              BNE   LBLA
021A 60                 RTS            END!--DISPLAY "W".
021B A9 FF     DEAL     LDA   #$FF     FILL CARBUF WITH
021D A2 33              LDX   #$33      ILLEGAL CARD #$FF
021F 95 B0     LBLB     STA   $00B0,X
0221 CA                 DEX
0222 10 FB              BPL   LBLB
0224 A0 33              LDY   #$33
0226 AD 1E A4  RAND     LDA   RIOTTI   6532 RIOT TIMER
0229 6D 04 A0           ADC   VIATI    6522 VIA TIMER
022C 29 3F     FXCARD   AND   #$3F     GENERATE PSEUDO-
022E 85 F8              STA   $00F8     RANDOM
0230 29 0F              AND   #$0F       TRIO-DECIMAL
0232 C9 0D              CMP   #$0D        CARDS
0234 B0 F0              BCS   RAND
0236 A5 F8              LDA   $00F8
0238 A2 33     FILLPK   LDX   #$33     CARD ALRFADY
023A D5 B0     LBLC     CMP   $00B0,X   IN PACK?
023C F0 E8              BEQ   RAND     TRY AGAIN
023E CA                 DEX
023F 10 F9              BPL   LBLC
0241 99 B0 00           STA   $00B0,Y CARDS INTO CARBUF
0244 88                 DEY
0245 10 DF              BPL   RAND
0247 20 8B 03  ARRHAN   JSR   SEQUEN   HANDS INTO
024A A0 0C     EXHAN    LDY   #$0C      BRIDGE SEQUENCE
024C B1 E4     LBLD     LDA   ($00E4),Y EXAMINE HAND--
024E 48                 PHA
024F 20 83 03           JSR   HIDIG    EXAMINE
0252 F6 F0     SULEN    INC   $00F0,X   HIGH ORDER DIGIT
0254 68                 PLA            INC SUIT LENGTHS
0255 29 0F     HIPTS    AND   #$0F      & LOW ORDER DIGIT
0257 38                 SEC
0258 E9 08              SBC   #$08     SUBTRACT 8 POINTS
025A 30 07              BMI   LBLE     POSITIVE?
025C 18                 CLC
025D 65 EE              ADC   $00EE    SUM INTO HIGH-CARD
025F 85 EE              STA   $00EE     POINT LOCATION
0261 85 EF              STA   $00EF    COPY INTO TOTAL
0263 88        LBLE     DEY             POINT LOCATION
0264 10 E6              BPL   LBLD
0266 A2 03     SUPTS    LDX   #$03     # OF CARDS IN SUIT
0268 38        LBLF     SEC             (S)
0269 B5 F0              LDA   $00F0,X
026B E9 04              SBC   #$04     SUBTRACT 4 FROM
026D 30 05              BMI   DUPLIC    LENGTH. POSITIVE?
026F 18                 CLC
0270 65 EF              ADC   $00EF    ADD TO TOTAL POINTS
0272 85 EF     TOTPTS   STA   $00EF
```

```
0274 B5 F0      DUPLIC  LDA   $00F0,X COPY SUIT LENGTHS
0276 95 F4              STA   $00F4,X  INTO $F4 TO $F7
0278 CA                 DEX             FOR STRONGEST
0279 10 ED              BPL   LBLF       SUIT
027B A0 0C      DIHAN   LDY   #$0C    DISPLAY CARDS
027D 98         LBLG    TYA            OF HAND
027E 48                 PHA
027F B1 E4              LDA   ($00E4),Y
0281 20 07 03           JSR   DIHANA
0284 68                 PLA           DISPLAY CARD
0285 A8                 TAY            PAUSE & BEEP
0286 88                 DEY
0287 10 F4              BPL   LBLG
0289 A9 F4      CHOOSU  LDA   #$F4    SET UP CLUBS
028B 85 E7              STA   $00E7    INDIRECT ADDRESSES
028D 85 E9              STA   $00E9
028F A9 F8              LDA   #$F8    --ALSO SPADES+1
0291 85 EB              STA   $00EB
0293 20 8B 03           JSR   SEQUEN  ARRANGE SUITS
0296 A5 F7              LDA   $00F7   IN ORDER
0298 A2 03              LDX   #$03    GET LONGEST AND
029A D5 F0      LBLH    CMP   $00F0,X  STRONGEST SUIT
029C F0 03              BEQ   LBLI
029E CA                 DEX
029F 10 F9              BPL   LBLH
02A1 86 F8      LBLI    STX   $00F8   AND PUT INTO $F8
02A3 A9 DB              LDA   #$DB    PUT "2." INTO
02A5 8D 44 A6           STA   DISBUF  +04 D+4
02A8 A9 39              LDA   #$39    PUT "C" INTO
02AA 8D 45 A6           STA   DISBUF  +05 D+5
02AD A5 EF      GOBID   LDA   $00EF   START BIDDING
02AF C9 19              CMP   #$19    T=25 POINTS OR MORE
02B1 90 02              BCC   LBLJ     ELSE CARRY ON
02B3 B0 3C              BCS   FINA    YES? FORCE TO FINA
02B5 C9 16      LBLJ    CMP   #$16    T=22 POINTS OR MORE?
02B7 90 6E              BCC   CONBID   ELSE GOT TO CONBID
02B9 A5 EE      NOTRUM  LDA   $00EE   NOTRUMP CALL?
02BB C9 15              CMP   #$15    H=21 POINTS?
02BD F0 2A              BEQ   FILLSU  ONLY 21! TO FILLSU
02BF C9 0F              CMP   #$0F    H=15 POINTS?
02C1 F0 26              BEQ   FILLSU  ONLY 15! TO FILLSU
02C3 A2 03              LDX   #$03
02C5 B5 F0      LBLK    LDA   $00F0.X
02C7 C9 02              CMP   #$02    SINGLETON OR VOID?
02C9 90 11              BCC   LONGSU
02CB D0 02              BNE   LBLL     ELSE CONTINUE
02CD E6 ED              INC   $00ED
02CF CA         LBLL    DEX
02D0 10 F3              BPL   LBLK    CHECK ALL 4 SUITS
02D2 A5 ED              LDA   $00ED
02D4 C9 02              CMP   #$02    2 DOUBLETONS?
02D6 B0 04              BCS   LONGSU  YES? TO LONGSU
02D8 A9 D4              LDA   #$D4    "N" INTO D+5
02DA D0 12              BNE   FIN     FORCE BRANCH TO FIN
02DC A5 EF      LONGSU  LDA   $00EF   DISTRIBUTION POINTS?
02DE C5 EE              CMP   $00EE    I.E. 5 CARD SUIT
02E0 D0 07              BNE   FILLSU   THEN GOTO FILLSU
02E2 A9 86      FILLAC  LDA   #$86      ELSE
02E4 8D 44 A6           STA   DISBUF  +04    "1" INTO D+4
02E7 D0 08              BNE   FINA       & FORCE TO FINA
02E9 A6 F8      FILLSU  LDX   $00F8   TAKE LONGEST
02EB BD C0 03           LDA   SUIT,X   & STRONGEST SUIT
02EE 8D 45 A6   FIN     STA   DISBUF  +05    & PUT INTO D+5
02F1 48         FINA    PHA
02F2 A0 80              LDY   #$80    DELAY ABOUT 10 SECS.
```

(continued)

you time to make North's bid. SYM's verdict then beeps onto the display. Did you call correctly? If North makes a valid bid, the round stops there, since a short routine like this cannot cope with overcalls and responses. If, however, North passes, ".E" is displayed on the left of the display and the process is repeated for East's hand, followed by South's and finally West's.

Before you start bidding you will have to ascertain SYM's conventions in advance, as per recognized bridge practice. Regrettably, its criteria are mechanical and pretty limited, but then some human bridge players are no better. High card points are summed as usual, with ace counting as 4; king, 3; queen, 2; and jack, 1. Distribution points are added by incrementing for every card in each suit above four. This is possibly no more inaccurate than the method of counting doubletons, singletons and voids, which is also partially performed in this routine, but only used for testing no-trumps bidding.

Above 24 total points automatically produces a strong two club call, and generally below 13 points a no-bid. Between 22 to 24 points and between 16 to 18 points, two no-trump and one no-trump calls are examined. The suit will then be no-trumps, providing that the high card points alone are not less than 22 or 16 points respectively, and the hand is balanced in the sense that there are no voids or singletons, and not more than one doubleton. This program does not examine individual suit stoppers. If no-trumps is eliminated, the longest and highest suit is chosen at the appropriate two or one level. If no suit longer than four cards is present, the bid will be a conventional one club, and a strong one club is bid with any hand possessing between 19 and 21 points. The usual element of confusion exists, in that any club call may possibly also be an ordinary genuine long club suit call. If no proper bid is valid, the routine tries one possible pre-emptive bid only. With a seven card suit or longer, and not less than 10 points including distribution, the bid will be three of the longest suit. Failing this, SYM will pass and no-bid with "P.S.".

To adapt for KIM, the upper zero-page storage locations can easily be relocated. The special SYM monitor routines and peripheral addresses have been underlined on the listing and subroutines marked with asterisks. These must be replaced by their KIM or other microprocessor equivalents. The program utilizes both X and Y indexed indirect addressing and can be relocated anywhere else in memory. However, I have somewhat unethically cut out a number of bytes in DISBUF and DUMPT routines, and a couple of minor alterations will be necessary if the program is moved to considerably higher memory. DUMPT is a subroutine in the new revised SYM monitor, which is utilized here to give a cheap 3 byte fixed delay of about 10 seconds. If a different delay is desired, store a byte into TAPDEL at location $A630 before the JSR to DUMPT:— 01 gives ? 4 seconds through to FF, which gives 7 1/4 minutes delay. Alternatively, this can be replaced by a standard delay routine. DISBUF in SYM's monitor System—RAM occupies $A640 = D + O to $A645 = D + 5, the former storing the extreme left hand character to be displayed, and the latter the right hand one. Three of these six locations are used for the display, one is blanked out with #00, and D+0 and D+2 are overwritten by the initial GO/200 command. When relocating to high memory on SYM, the necessity for any alteration to these two routines can be obviated if desired by simply adding a JMP to start address command at, say, location $300.

TV controls the period of DELAY logarithmically, and location $031C can be altered for quicker or slower display of the cards. KIM owners note that the SYM monitor routine DELAY includes SCAND. It transfers the six segment codes previously stored in DISBUF to the 7-segment display, and retains this display for the period set into TV. The deck of four hands is dealt into CARBUF zero-page locations $00B0 to $00E3. Locations $00E4 to $00F8 are used for indirect base addresses, SEQUEN locations, flags for points, suits, inbalance, long-suit, etc. The cards themselves do not get disturbed during the execution of the program and could be used later for continuing and extending the game. The display codes in the tables at the end have been chosen to be as clear and unambiguous as possible, within the obvious limitations of 7-segments. One or two are a bit weird and need getting used to.

SYM has a useful Verify Checksum routine, amongst others, directly available at its keypad. In order to check that no byte has been keyed in incorrectly or subsequently altered, either before or after running the pro-

```
02F4 20 87 8E          JSR  DUMPT     NEW MONITOR (MON 1.1)
02F7 20 72 89          JSR  BEEP      1 ONBOARD BEEP
02FA 68                PLA
02FB C9 EE             CMP  #$EE      "PASS?"--CONTINUE
02FD D0 02             BNE  LBLM      ANY BID?--STOP
02FF F0 1A             BEQ  DIBEEP    FORCE TO DIBEEP
0301 20 1B 03  LBLM    JSR  DIBEEP    JSR DIBEEP
0304 4C 00 80          JMP  MONITR    MONITOR COLD--END!
0307 48        DIHANA  PHA            SHOW CARDS OF HAND
0308 29 0F             AND  #$0F      LOW ORDER DIGIT
030A AA                TAX
030B BD C4 03          LDA  VALUE,X   CARD FACE VALUE
030E 8D 44 A6          STA  DISBUF    +04  INTO D+4
0311 68                PLA
0312 20 83 03          JSR  HIDIG     --GET HIGH DIGIT
0315 BD C0 03          LDA  SUIT,X    CARD SUIT
0318 8D 45 A6          STA  DISBUF    +05 INTO D+5
031B A9 0B     DIBEEP  LDA  #$0B      DISPLAY FOR
031D 8D 56 A6          STA  TV         3 SECONDS
0320 20 5A 83          JSR  DELAY       APPROX.
0323 20 72 89          JSR  BEEP        &BEEP ONCE
0326 60                RTS
0327 A9 86     CONBID  LDA  #$86      CONTINUE BIDDING
0329 8D 44 A6          STA  DISBUF    +04 "1." INTO D+4
032C A5 EF             LDA  $00EF     TOTAL POINTS AGAIN
032E C9 13             CMP  #$13      19 POINTS OR MORE?
0330 90 02             BCC  LBLN       ELSE CONTINUE
0332 B0 BD             BCS  FINA      FORCE BRANCH TO FINA
0334 C9 10     LBLN    CMP  #$10      16 POINTS OR MORE?
0336 90 03             BCC  LBLO       ELSE CONTINUE
0338 4C B9 02          JMP  NOTRUM
033B C9 0D     LBLO    CMP  #$0D      13 POINTS OR MORE?
033D 90 02             BCC  PREMPT     ELSE CONTINUE
033F B0 9B             BCS  LONGSU    FORCE TO LONGSU
0341 C9 0A     PREMPT  CMP  #$0A      10 POINTS OR MORE?
0343 90 0D             BCC  NOBID      ELSE NOBID
0345 A5 F7             LDA  $00F7     7 CARD SUIT OR MORE?
0347 C9 07             CMP  #$07
0349 90 07             BCC  NOBID
034B A9 CF             LDA  #$CF      "3." INTO D+4
034D 8D 44 A6          STA  DISBUF    +04
0350 D0 97             BNE  FILLSU    FORCE TO FILLSU
0352 A9 F3     NOBID   LDA  #$F3      "P." INTO D+4
0354 8D 44 A6          STA  DISBUF    +04
0357 A9 ED             LDA  #$ED      "S" INTO D+5
0359 D0 93             BNE  FIN       FORCE TO FIN
035B A9 00     INIT    LDA  #$00      SUBROUTINE:
035D 8D 43 A6          STA  DISBUF    +03 FILL WITH 0--
0360 A2 13             LDX  #$13       D+3 & STORAGE
0362 95 E4     LBLP    STA  $00E4,X    LOCATIONS
0364 CA                DEX             $E4 TO $F8
0365 10 FB             BPL  LBLP
0367 B9 BC 03          LDA  SEGMNT,Y "N","E","S","W"
036A 8D 41 A6          STA  DISBUF    +01  INTO D+1
036D B9 B4 03          LDA  STRTAD,Y
0370 85 E4             STA  $00E4
0372 85 E7             STA  $00E7     ADDRESSES FOR HANDS
0374 85 E9             STA  $00E9
0376 B9 B8 03          LDA  ENDAD,Y
0379 85 EB             STA  $00EB
037B 60                RTS            S.R..ROUND E,S, & W
037C 20 5B 03  INITA   JSR  INIT
037F 20 47 02          JSR  ARRHAN
0382 60                RTS
0383 EA        HIDIG   NOP            SUBROUTINE:
```

```
0384 EA              NOP                PUT HIGH ORDER
0385 4A              LSR    A           DIGIT INTO
0386 4A              LSR    A           X INDEX REGISTER
0387 4A              LSR    A           (BOTH NOP'S ARE
0388 4A              LSR    A            SUPERFLUOUS!)
0389 AA              TAX
038A 60              RTS
038B A2 00  SEQUEN   LDX    #$00       SUBROUTINE:
038D A1 E7  LBLQ     LDA    ($00E7,X)  REARRANGE IN
038F C1 E9           CMP    ($00E9,X)  ASCENDING ORDER
0391 90 0C           BCC    LBLR        CONTENTS OF ANY
0393 A1 E7           LDA    ($00E7,X)  CONTIGUOUS
0395 85 E6           STA    $00E6      ZERO-PAGE
0397 A1 E9           LDA    ($00E9,X)  MEMORY BLOCK...
0399 81 E7           STA    ($00E7,X)
039B A5 E6           LDA    $00E6
039D 81 E9           STA    ($00E9,X)  ...START & STOP+1
039F E6 E9  LBLR     INC    $00E9       INDEXED INDIRECT
03A1 A5 E9           LDA    $00E9       ADDRESSES
03A3 C5 EB           CMP    $00EB       $E7 TO $EC
03A5 D0 E6           BNE    LBLQ        ..$E6 IS A
03A7 E6 E7           INC    $00E7       WORK LOCATION
03A9 A5 E7           LDA    $00E7
03AB 85 E9           STA    $00E9
03AD A5 E9           LDA    $00E9
03AF C5 EB           CMP    $00EB
03B1 D0 DA           BNE    LBLQ
03B3 60              RTS

03B4 D7     STRTAD   =      $D7     W  START ADDRESSES
03B5 CA              =      $CA     S
03B6 BD              =      $BD     E
03B7 B0              =      $B0     N

03B8 E4     ENDAD    =      $E4     W  END+1 ADDRESSES
03B9 D7              =      $D7     S
03BA CA              =      $CA     E
03BB BD              =      $BD     N

            DISPLAY SEGMENT TABLES

03BC 3A     SEGMNT   =      $3A     W  (L')
03BD 6D              =      $6D     S
03BE 79              =      $79     E
03BF 54              =      $54     n

03C0 39     SUIT     =      $39     C
03C1 5E              =      $5E     d
03C2 76              =      $76     H
03C3 6D              =      $6D     S

03C4 DB     VALUE    =      $DB     2.
03C5 CF              =      $CF     3.
03C6 E6              =      $E6     4.
03C7 AD              =      $AD     5. (5.)
03C8 FD              =      $FD     6.
03C9 87              =      $87     7.
03CA FF              =      $FF     8.
03CB E7              =      $E7     9.
03CC F8              =      $F8     10 (E.)
03CD 9E              =      $9E     J.
03CE EB              =      $EB     Q. (Q.)
03CF F0              =      $F0     K. (F.)
03D0 F7              =      $F7
```

gram, the correct 4 digit checksum from location $0200 to $03D0 is #F590.

Shortly after completing this article, I received SYM's mini 2K Symbolic Assembler and have just got it up and running. Now for the first time I am experiencing the pros and cons of assembly language programming. To date I have been entirely confined to purely hand assembled machine code. Therefore, I merely put this program through my rudimentary SYM disassembler to produce the accompanying 6502 format listing, and added some pseudo-assembler labels and plenty of comments for elucidation.

The key routine of this program is FXCARD. It takes a stream of pseudorandom hexadecimals and alters them, where necessary, to the 52 unique triodecimal integers between #00 and #3C. The low order digit indicates the card's face value from two to ace. The high order digit indicates the suit in bridge strength order from clubs to spades. For example, the numbers 3C, 38, 32, 2A, 29, 25, 21, 1C, 17, 15, 0C, 0B, 06 represent a maximum one no-trump hand;—Spades—A, T, 4; Hearts—Q, J, 7, 3; Diamonds—A, 9, 7; Clubs—A, K, 8. FXCARD simply ANDS out the hexadecimals with 3F, and filters out those ending in D, E and F. This system later enables simple examination of each card by merely ANDing plus right shifts. SULEN determines suit lengths (S), by incrementing four counters for cards with identical high order digits. HIPTS subtracts #8 from the low order digits and sums all positive results giving the high-card point count (H). Afterwards, the distribution points for all suits longer than 4 cards are added to provide the total points (T) in TOTPTS. DIHAN transfers the digits to the Y index register, and the two appropriate alphanumeric segment codes for each card are put into DISBUF, using indexed addressing. The cards of one hand are then displayed, accompanied by pauses and beeps. Arranging the cards of a hand into correct bridge sequence is performed conveniently by the standard subroutine SEQUEN, which also serves CHOOSU to determine the longest and strongest suit for declaring.

The pseudorandom # generation is very crude. SYM has 7 timers built into its 4 VIA's, and here one of the 6522's is used together with the 6532 timer. The statistics are probably highly questionable, but nevertheless RAND, at the cost of 6 bytes, seems to give the same sort of random hands I get when I actually play bridge. A more sophisticated routine can always be substituted if desired. FILLPK is doubtless very wasteful on time, but quite frugal on memory. It merely goes round the previous cards selected and rejects any valid trio-decimal smaller than 3D, which has been selected previously. Despite its time inefficiency, it only takes a fraction of a second to shuffle and deal out the whole deck. The rest of the program, apart from the deliberately built in delay and pauses, is virtually instantaneous.

The bidding program runs from maximum points downwards, finishing with no-bid in default of any valid bid. Routines for no-trumps, long suit, clubs etc., are used more than once. The bidding program should be clear from the comments in the listing, and the previously mentioned bidding criteria. The only aspect which possibly warrants any comment is the no-trump bidding.

On the possibly debatable hypothesis that a valid no-trump call takes precedence over others, the criteria adopted for no-trumps are the three that I mentioned earlier. There is at least partial overlapping with some of the other routines essential to the program, and therefore several different possibilities of programming for no-trumps. The method I have chosen is not necessarily the shortest or best. In addition to the short suit tests for voids and singletons and doubletons, I have employed a test for 15 or 21 high card points. No additional tests are necessary since 14 high-card points, for example, together with the mandatory minimum of 16 total points, indicate at least 2 distribution points, and hence one 6-card suit, or two five-card suits etc. It should be evident that all of these are inconsistent with the short suit criteria for no-trumps, and hence would in any case be invalidated later on, on that count. Incidentally, once any distribution points are discovered, there must obviously be at least one biddable suit present. It is then superfluous to check for a one club call, and so the routine skips LONGSU and branches direct to FILLSU. LONGSU is the general routine which compares the total points with the high-card points. If these are equal, there are no distribution points, no long suit, and one club is called. If they are unequal, FILLSU transfers the longest and strongest suit to DISBUF.

SYM has not been programmed to bid the ACOL or Precision systems or the like, but simply a homely hybrid of amateur conventions. Experienced bridge players may be particularly shocked that SYM calls one club with any good 19 to 21 point hand, and also with a large range of hands strong and weak containing no long suit, in addition to the regular genuine one club hand. But all of this can be modified in the programming to conform with *any* conventions according to personal tastes and individual requirements.

If desired, the program can be easily be altered to continue round all four hands after valid bids, to redisplay a hand, to bid preselected or problem hands, and a scoring and timing system could be added to increase tension and interest. If an extra page or so of RAM is available, SYM's onboard 'scope interface can be exploited to display each complete hand much more aesthetically, incorporating conventional card graphics for the four suits if so desired.

This routine certainly does not cover all eventualities; I don't know whether there are any conventions which do. The other day SYM's North "picked up", Spades—void; Hearts—A,K,Q,J,8,7,5,4,2; Diamonds—void; Clubs—T,9,8,4:—and naturally "1.H." appeared on the display. My shouting that this was an obvious four or five hearts, or at the very least one strong club, in order to keep the bidding open for one round, was, of course, a waste of breath. At least nobody screamed back! To obviate similar anomalies, it might be worth adding bonus distribution points on an increasing sliding scale for 8-card suits and longer.

I should like to take this opportunity to thank the several experienced SYMmers in the United States, England, and New Zealand who have assisted and encouraged me by corresponding over the past year. I should also like to express my appreciation to the authors of *The First Book of KIM.* Apart from generally acting as a stimulating catalyst and learning tool, I have also modified and built upon basic ideas contained in their Blackjack, Card Dealer and Sort routines.

Len Green was born and educated in London, has travelled extensively in Europe, and now resides in Israel, where he has lived for nearly 30 years. He enjoys experimenting with his SYM, which he purchased in 1978, and he still considers himself to be a "newcomer" to hobbyist computing.

MICRO

MICRO PET Vet

By Loren Wright

Update on the VIC

I had expected to have a VIC prototype in time for this month's column. I am assured by Commodore, however, that I will have one in time for February's PET Vet. Meanwhile, I have learned a few additional details. The price is $299.95, which includes 3.8K of user RAM and a cable for a monitor, but not the TV modulator. The primary method of memory expansion is with a plug-in cartridge, containing 3K of RAM and sockets for two 2716-type ROMs. These cartridges will be available to independent software houses, so that they can produce their own plug-in program cartridges for the VIC. The second method of expanding is through a master control board, which can accept 3K, 8K, or 16K RAM expansion modules, other plug-in programs, or special cards, such as a planned IEEE-488 interface.

The VIC, unlike the PET and CBM machines, has a serial bus, and for this Commodore will sell a serial version of the CBM 2031 single floppy disk drive, and a serial printer. Other peripherals for the VIC will be offered later, but to use any of the existing parallel devices, the IEEE-488 card is required.

The official release of the VIC will coincide with the Consumer Electronics Show in Las Vegas in January, 1981.

Finding BASIC Variables

Finding a BASIC variable is quite simple using a routine that already exists in the PET's BASIC ROMs. Two zero-page locations are used to store the name of the variable, and the routine returns the address of that variable in the next two locations. The two addresses for the name are set up with the ASCII codes for the characters. If the name doesn't have a second character, then 0 is entered. In addition, though, string variables have $80 ORed into the second address, and integer variables have $80 ORed into both locations.

Examples:

Variable	VARNAM	VARNAM+1
X	$58	$00
L$	$4C	$80
AB	$41	$42
D1%	$C4	$B1

Floating point variables are stored in a special 7-byte memory format—the name in two bytes, as above, the exponent, and four mantissa bytes. Integer variables reserve the same seven bytes, but only use the first four—two for the name and two for the integer in high, low order. Strings are actually stored in high memory, but in low memory seven bytes are reserved, of which five are used. The name occupies two bytes, followed by the length of the string, and the starting address (low, high) of the actual string in high memory.

BASIC ROM Version

	2.0	3.0	4.0
VARNAM	$94	$42	$42
VARNAM+1	$95	$43	$43
VRADLO	$96	$44	$44
VRADHI	$97	$45	$45
Primary FAC	$B0-$B5	$5E-$63	$5E-$63
FNDVAR	$CFD7	$CFC9	$C187
MEMFAC	$DA74	$DAAE	$CCD8
FACINT	$D0A7	$D09A	$C2EA

If VARNAM and VARNAM+1 are set up properly, a call to FNDVAR will return with the address of that variable in VRADLO,VRADHI. This address is also in the accumulator (low) and Y register (high), so in the case of a floating point variable, an immediate call of MEMFAC will move the number to the primary FAC (floating point accumulator). The conversion from memory format to FAC format is made on the way. A call to FACINT will convert the FAC to an integer in the fourth and fifth bytes ($B3,$B4 or $61,$62 in high,low order).

The address indicated by VARADR,VARADR+1 is actually the location of the first byte after the name portion of the entry. With an integer variable, this address will contain the high byte of the integer value, but with a string variable, it will contain the length of the string. The next two bytes contain the actual address of the string in high memory. The following routine will copy the found string to memory starting at address XXXX (whatever you choose). Suggested locations for ADLO,ADHI are $32,$33 (BASIC 3.0 and 4.0) and $71,$72 (2.0).

```
     LDY #$02
     LDA (VRADLO),Y   The third byte
     STA ADHI
     DEY
     LDA (VRADLO),Y   The second
     STA ADLO
     DEY
     LDA (VRADLO),Y   The first—length
     TAY
LOOP LDA (ADLO),Y
     STA XXXX,Y
     DEY
     BNE LOOP
```

Now that you know how to find BASIC variables, what do you do with them? Well, a typical application is in a plotting program where the only variables of concern might be X and Y. The BASIC program might have a FOR...NEXT loop on X, with Y calculated within the loop, followed by a SYS call to your routine which finds X and Y and then processes them for plotting on the screen.

If you know where the variables are, that means you can change them (carefully) without disturbing BASIC. All kinds of fancy string manipulations come to mind. Not only can the characters be changed, but also the lengths, names, and locations.

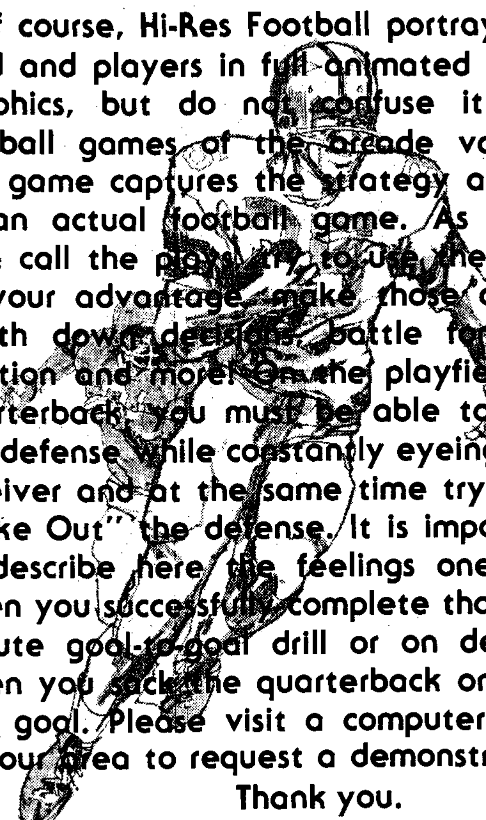
I have two additions to my column on documentation (MICRO 29:39)—*Library of PET Subroutines* and *The PET Revealed*, both by Nick Hampshire, $19.95 each. These have been available in the United Kingdom and by mail for quite a while, but now Commodore has "officially approved" them and plans to sell them through its dealers in the U.S.

The PET Revealed is a well-organized collection of interesting and important information on the inner workings of the PET. The five general sections are: (1) The PET System Hardware, (2) The 6502 Microprocessor, (3) The PET Operating System, (4) The User Port, and (5) The IEEE Port and the 6520. Complete schematics, memory maps, BASIC subroutines, BASIC tokens, and lots of other charts and tables are included. Also included is a number of special application circuits and routines. This is a very useful book for anyone wanting to go beyond BASIC.

Library of PET Subroutines is a collection of common routines in BASIC and assembly language. A general-purpose screen handler, random-access disk file management, and several sort programs are examples. Anyone wishing to take on a major programming project, without starting from scratch, should appreciate this book.

ON-LINE SYSTEMS
PRESENTS
HI-RES FOOTBALL

By JAY SULLIVAN And KEN WILLIAMS



This is the football game you always thought your Apple was capable of, but no one seemed able to deliver!

Of course, Hi-Res Football portrays the field and players in full animated Hi-Res Graphics, but do not confuse it with football games of the arcade variety. This game captures the strategy aspects of an actual football game. As coach you: call the plays, try to use the clock to your advantage, make those critical fourth down decisions, battle for field position and more! On the playfield, as quarterback you must be able to read the defense while constantly eyeing your receiver and at the same time trying to "Fake Out" the defense. It is impossible to describe here the feelings one gets when you successfully complete that two minute goal-to-goal drill or on defense when you sack the quarterback on third and goal. Please visit a computer store in your area to request a demonstration.

Thank you.

Runs on any 48K Apple II or II Plus, DOS 3.2 or 3.3. Available now for $39.95 on disk from your local computer store or you may order directly from:

ON-LINE SYSTEMS
36575 Mudge Ranch Road
Coarsegold, CA 93614
209-683-6858

ORDERS MAY BE CHECK, VISA, MASTER CHARGE OR C.O.D.

Find Your Way Around The New Apple® DOS With The Dakin5® Programming Aids 3.3©

Dakin5 Corporation, a Colorado software house, is making available to the public 12 utility programs on one 16 sector diskette, utilizing the new Apple DOS 3.3, which provides 23% more storage.

All of the Dakin5 Programming Aids 3.3 programs are also compatible with the Corvus Disk Drive system.

Features

- Remove REM statements, unreferenced (dead) code, and compress code to increase program speed and save memory and disk space.
- Copy any file or program from one diskette to another. Only the name is needed.
- Print or display a line cross reference and variable name cross reference.
- Print or display all or selected records from a text file.
- Display any sector of a given file or program, and then update any data within that sector, or specify the sector you wish to update, such as directory sectors and sectors occupied by DOS.
- Create, print and modify your own text and Exec files.
- Perform 20-digit arithmetic.
- Copy a diskette without DOS; initialize without DOS; verify source diskette; verify copied data is the same as the original.
- Use a powerful data entry routine that handles both string and numeric data.

Plus Many More Utility Programs for Sophisticated Programmers

Many of these utility programs have been developed and tested for in-house use while producing The Controller™ business package for Apple Computer Inc.

Each programming aids package includes a program diskette and very complete documentation, all attractively packaged in a padded, blue print vinyl 3-hole notebook with silver lettering. An identifying tab separates each program for convenient reference.

See your Apple dealer or contact Dakin5 Corporation, P.O. Box 21187, Denver, Colo. 80221. Telephone: 800-525-0463. VISA or MC welcome.

Apple is a registered trademark of Apple Computer Inc. The Controller is a registered trademark of Dakin5 Corporation.

DAKIN5 CORPORATION

Decision Systems

Decision Systems
P.O. Box 13006
Denton, TX 76203

SOFTWARE FOR THE APPLE II*

ISAM-DS is an integrated set of Applesoft routines that gives indexed file capabilities to your **BASIC** programs. Retrieve by key, partial key or sequentially. Space from deleted records is automatically reused. Capabilities and performance that match products costing twice as much.
$50 Disk, Applesoft.

PBASIC-DS is a sophisticated preprocessor for structured **BASIC**. Use advanced logic constructs such as **IF...ELSE...**, **CASE**, **SELECT**, and many more. Develop programs for Integer or Applesoft. Enjoy the power of structured logic at a fraction of the cost of **PASCAL**.
$35. Disk, Applesoft (48K, ROM or Language Card).

DSA—DS is a dis-assembler for 6502 code. Now you can easily dis-assemble any machine language program for the Apple and use the dis-assembled code directly as input to your assembler. Dis-assembles instructions and data. Produces code compatible with the S-C Assembler (version 4.0), Apple's Toolkit assembler and others.
$25 Disk, Applesoft (32K, ROM or Language Card).

FORM-DS is a complete system for the definition of input and output froms. **FORM-DS** supplies the automatic checking of numeric input for acceptable range of values, automatic formatting of numeric output, and many more features.
$25 Disk, Applesoft (32K, ROM or Language Card).

UTIL-DS is a set of routines for use with Applesoft to format numeric output, selectively clear variables (Applesoft's **CLEAR** gets everything), improve error handling, and interface machine language with Applesoft programs. Includes a special load routine for placing machine language routines underneath Applesoft programs.
$25 Disk, Applesoft.

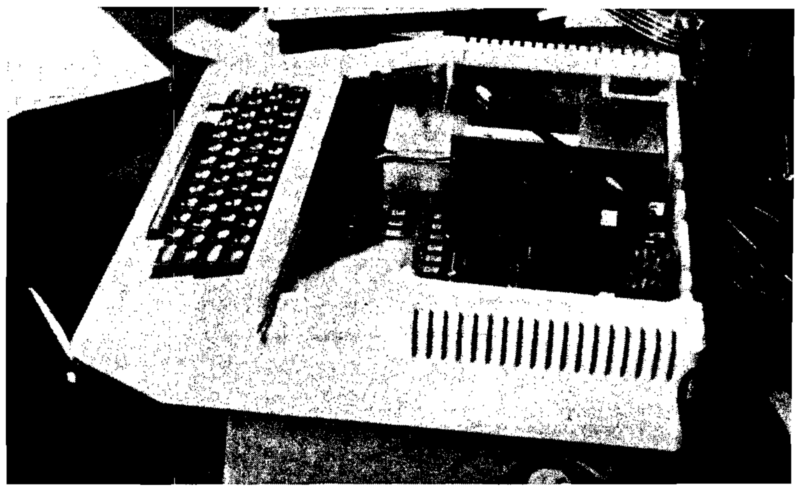
SPEED-DS is a routine to modify the statement linkage in an Applesoft program to speed its execution. Improvements of 5-20% are common. As a bonus, **SPEED-DS** includes machine language routines to speed string handling and reduce the need for garbage clean-up. Author: Lee Meador.
$15 Disk, Applesoft (32K, ROM or Language Card).

(Add $4.00 for Foreign Mail)

***Apple II is a registered trademark of the Apple Computer Co**

Make a Clear Plastic Cover For Your Apple

E.J. Neiburger, D.D.S.
1000 North Avenue
Waukegan, Illinois 60085



Your APPLE has a cover which, if removed, will allow you access to the boards and chips located on the mother board. What I describe below is a simple project that will create a clear plastic, duplicate cover that will protect your APPLE's innards—yet allow the curious to look in.

Buy a clear sheet of ¼ thick window plastic (for example, Lexan) from your local hardware store. You will need an 11¾- by 13-inch piece. Place the plastic on two 12-inch sections of 1½-inch square wood so that 5 inches of the plastic extends over the edge (see figures 1 and 2).

Heat evenly with a blowtorch until the plastic begins to sag. Push the 5-inch section down until it contacts the table surface and hold until cool. Heat the plastic gradually to avoid bubbling or yellowing.

Glue two plastic blocks (1- X ½- X ¼-inch) to the back edge of the plastic, approximately 2 inches from either side. Attach self-adhesive fasteners, as seen on the APPLE cover and ....

Watch while you are computing!

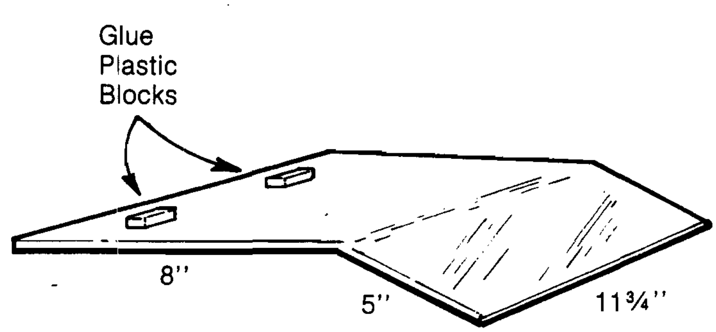


Figure 1

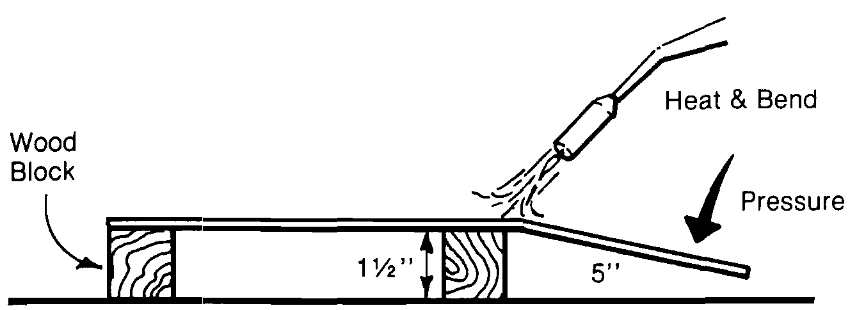
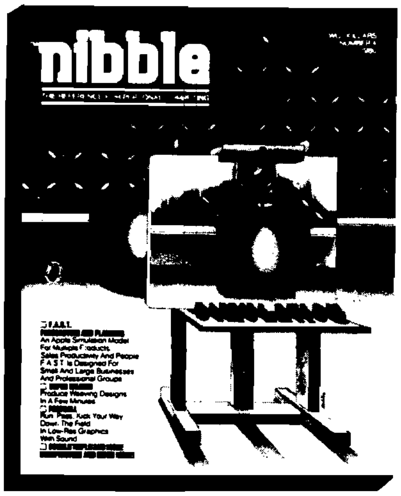


Figure 2

Ellis "Skip" Neiburger is a practicing dentist, Editor of the *Dental Computer Newsletter*, Contributing Editor for the *Physician's Micro Computer* report and author of several computer science articles. He is director of Andent Inc., a computer applications firm. His interests involve the use of microcomputers in medical-dental treatment, robot systems, and practical real-world applications.

MICRO

"NIBBLE® IS TERRIFIC" (For Your Apple)



NIBBLE IS: *The Reference for Apple computing!*

NIBBLE IS: One of the Fastest Growing new Magazines in the Personal Computing Field.

NIBBLE IS: Providing Comprehensive, Useful and Instructive Programs for the Home, Small Business, and Entertainment.

NIBBLE IS: A Reference to Graphics, Games, Systems Programming Tips, Product News and Reviews, Hardware Construction Projects, and a host of other features.

NIBBLE IS: A magazine suitable for both the Beginner and the Advanced Programmer.

Each issue of NIBBLE features significant new Programs of Commercial Quality. Here's what some of our Readers say:

- *"Certainly the best magazine on the Apple II"*
- *"Programs remarkably easy to enter"*
- *"Stimulating and Informative; So much so that this is the first computer magazine I've subscribed to!"*
- *"Impressed with the quality and content."*
- *"NIBBLE IS TERRIFIC!"*

In coming issues, look for:

☐ Numeric Keypad Construction Lab ☐ Assembly Language Programming Column
☐ Pascal Programming Column ☐ Data Base Programs for Home and Business
☐ Personal Investment Analysis ☐ Electronic Secretary for Time Management
☐ The GIZMO Business Simulation Game

And many many more!

NIBBLE is focused completely on the Apple Computer systems.

Buy NIBBLE through your local Apple Dealer or subscribe now with the coupon below.

Try a NIBBLE!

nibble No. 4

Box 325, Lincoln, MA. 01773 (617) 259-9710

I'll try nibble!
Enclosed is my $17.50 (for one year).
☐ **check** ☐ **money order**

(Please allow 4 to 6 weeks for delivery of 1st issue)
BACK ISSUES of NIBBLE are available for $2.00 + .50 postage and handling.

Name ____________________

Address ____________________

City ____________________

State ____________ Zip ________

NOTE:
First Class or Air Mail is required for all APO, FPO and all foreign addresses with the following additional amounts.
—USA, Canada, Mexico, APO, FPO $7.50
—Central and South America $9.00
—Europe $12.00
—Asia and elsewhere $15.00

Searching String Arrays

This machine language program makes searching a large string array considerably faster and easier.

Gary B. Little
#101-2044 West Third Avenue
Vancouver, British Columbia
Canada V6J 1L5

Have you Apple users ever wanted to search through a string array to see if it contains a particular phrase? If you have, it's probable that you have written a rather short loop routine in Applesoft to do this. However, if you have a few thousand comparisons to make, the Applesoft version may take an undesirable length of time to grind out the desired results.

A much faster search can be carried out on the Apple II by using a search routine written in 6502 assembly language. Such a program is shown below.

To understand exactly how the program works it is necessary to analyze the method by which the Apple stores variables in its memory. The details are found on page 137 of the *Applesoft II Basic Programming Reference Manual*. For a one-dimensional string array, the storage pattern is as follows:

NAME (2 bytes)
OFFSET pointer to next variable (2 bytes)
No. of dimensions (1 byte)
Size 1st dimension (2 bytes)
String$(0)—length (1 byte)
—address low (1 byte)
—address high (1 byte)

String$(N) (3 bytes)

N is the size of the 1st dimension. If the string array is the first array variable defined in a program, the memory location of the first byte of the trio of bytes, reserved for the Cth array variable, is given by PEEK(107) + 256*PEEK(108) + 7 + 3*C (where 0 =C =N). This is because the pointer to the beginning of the array space, and also to the beginning of the string array variable map, is found at $6B,$6C (107,108) and there are 7 + 3*C bytes before the three Cth array variable bytes.

If the phrase to be searched for (the search variable) is the first simple variable defined in a program, the memory location of the first byte of the three bytes reserved for the length and location of the string is given by PEEK(105) + 256*PEEK(106) + 2. This is because the pointer to the beginning of the simple variable space, and also to the beginning of the simple variable map, is found at $69,$6A (105,106). There are two bytes before the three variable bytes.

To carry out the search, it is simply necessary to compare the string pointed to by SV + 3,SV + 4 (where SV = PEEK(105) + 256*PEEK(106)) with the string pointed to by AV + 8 + 3*C,AV + 9 + 3*C (where AV = PEEK(107) + 256*PEEK(108) and C runs from 0 to N). This is precisely what is done in this assembly language routine.

```
                    LENS     EQU  $0        LENGTH OF SEARCH PHRASE
                    LENR     EQU  $1        LENGTH OF STRING ARRAY VARIABLE
                    SP       EQU  $6        POINTER TO SEARCH PHRASE
                    RP       EQU  $8        POINTER TO ARRAY VARIABLE TABLE
                    RL       EQU  $1A       POINTER TO ARRAY VARIABLE
                    NL       EQU  $1C       ENDING ARRAY NUMBER
                    CL       EQU  $1E       STARTING ARRAY NUMBER (AND COUNTER)
                    SAVE     EQU  $FF4A     SAVE REGISTERS
                    RESTORE  EQU  $FF3F     RESTORE REGISTERS
                             ORG  $300
0300: 20 4A FF               JSR  SAVE      SAVE REGISTERS
0303: A0 00         LOOP     LDY  #$00
0305: B1 08                  LDA  (RP),Y    GET LENGTH OF VARIABLE
0307: 85 01                  STA  LENR       AND STORE
0309: C8                     INY
030A: B1 08                  LDA  (RP),Y    GET POINTER (LO)
```
(continued)

```
030C: 85 1A           STA  RL          AND SAVE
030E: C8              INY
030F: B1 08           LDA  (RP),Y     GET POINTER (HI)
0311: 85 1B           STA  RL+1        AND SAVE
0313: A5 01           LDA  LENR       IF LENGTH OF SEARCH
0315: C5 00           CMP  LENS        PHRASE EXCEEDS LENGTH
0317: 30 0F           BMI  NOPE        OF VARIABLE, SEARCH FAILS
0319: A0 00           LDY  #$00
031B: B1 06    AGAIN  LDA  (SP),Y     COMPARE THE PHRASES
031D: D1 1A           CMP  (RL),Y      LETTER BY LETTER
031F: D0 07           BNE  NOPE       FAILS IF NOT EQUAL
0321: C8              INY
0322: C4 00           CPY  LENS
0324: F0 10           BEQ  RTS1       SUCCESS!
0326: D0 F3           BNE  AGAIN
0328: A5 1E    NOPE   LDA  CL         COMPARE COUNTER
032A: C5 1C           CMP  NL          TO ENDING ARRAY NUMBER
032C: D0 0B           BNE  LOOP1
032E: A5 1F           LDA  CL+1
0330: C5 1D           CMP  NL+1
0332: D0 05           BNE  LOOP1      DONE IF EQUAL
0334: E6 1F           INC  CL+1
0336: 4C 3F FF RTS1   JMP  RESTORE
0339: 18       LOOP1  CLC
033A: A5 08           LDA  RP         SET POINTER TO NEXT
033C: 69 03           ADC  #$03        TRIO OF ARRAY BYTES
033E: 90 02           BCC  N1
0340: E6 09           INC  RP+1
0342: 85 08    N1     STA  RP
0344: 18              CLC
0345: A5 1E           LDA  CL         INCREMENT COUNTER
0347: 69 01           ADC  #$01
0349: 90 02           BCC  N2
034B: E6 1F           INC  CL+1
034D: 85 1E    N2     STA  CL
034F: 38              SEC
0350: B0 B1           BCS  LOOP       CHECK NEXT ARRAY VARIABLE
```

The time savings that can be realized by using the routine can be seen by running the Applesoft demo program that is LISTed below. For example, an assembly language search of 2,000 string array variables takes only one second, whereas the same search done in Applesoft takes 19 seconds!

To use the search routine from within an Applesoft program, the following procedure must be followed:

1. POKE the length of, and the two pointers to, the search phrase into locations 0,6,7, respectively. This is done in line #210 of the demo program.
2. POKE the number of the array variable from which the search is to proceed ('C') in locations 30,31 (low,high). This is done in line #220.

```
100 S$ = "": REM MUST BE FIRST
     DEFINED SIMPLE VARIABLE
110 N = 2000: DIM R$(N): REM MUST
      BE FIRST DEFINED ARRAY
     VARIABLE
120  GOSUB 1000: REM LOAD SEARCH
     ROUTINE
130  DEF  FN MD(X) = X - 256 *  INT
     (X / 256)
140  TEXT : HOME : PRINT  TAB( 8)
     ;: INVERSE : PRINT "STRING A
     RRAY SEARCH DEMO": NORMAL
150  PRINT : PRINT "RANDOM STRING
     S:": PRINT
160  FOR I = 1 TO N:R$(I) =  CHR$
     (65 + 26 *  RND (1)) +  CHR$
     (65 + 26 *  RND (1)): PRINT
     R$(I);" ";: NEXT I: PRINT : PRINT
```

```
170  INPUT "ENTER SEARCH STRING:
     ";S$: PRINT
180 SV = AV:C = 1
190 SV =  PEEK (105) + 256 *  PEEK
     (106)
200 AV =  PEEK (107) + 256 *  PEEK
     (108)
210  POKE 0, PEEK (SV + 2): POKE
     6, PEEK (SV + 3): POKE 7, PEEK
     (SV + 4)
220  POKE 30, FN MD(C): POKE 31, INT
     (C / 256)
230  POKE 28, FN MD(N): POKE 29, INT
     (N / 256)
240  POKE 8, FN MD(AV + 7 + 3 * C
     ): POKE 9, INT ((AV + 7 + 3 *
     C) / 256)
250  CALL 768
260 C =  PEEK (30) + 256 *  PEEK
     (31)
270  IF C > N THEN 300
280  PRINT S$;" MATCHES #";C;" (P
     HRASE: ";R$(C);")"
290 C = C + 1: IF C <  = N THEN 1
     90
300  PRINT : PRINT "MACHINE LANGU
     AGE SEARCH COMPLETED"
310  PRINT : INPUT "PRESS 'RETURN
     ' FOR APPLESOFT SEARCH: ";A$
     : PRINT
320  FOR I = 1 TO N
330  IF S$ =  LEFT$ (R$(I), LEN (
     S$)) THEN  PRINT S$;" MATCHE
     S #";I;" (PHRASE: ";R$(I);")
     "
340  NEXT I: PRINT : PRINT "APPLE
     SOFT SEARCH COMPLETED": END

1000  FOR I = 768 TO 849: READ X:
      POKE I,X: NEXT I: RETURN
1010  DATA 32,74,255,160,0,177,8,
     133,1,200,177,8,133,26,200,1
     77,8,133,27,165,1,197,0,48,1
     5,160,0,177,6,209
1020  DATA 26,208,7,200,196,0,240
     ,16,208,243,165,30,197,28,20
     8,11,165,31,197,29,208,5,230
     ,31,76,63,255,24,165,8
1030  DATA 105,3,144,2,230,9,133,
     8,24,165,30,105,1,144,2,230,
     31,133,30,56,176,177
```

3. POKE the number of the array variable, at which the search is to end, ('N') in locations 28,29 (low,high). This is done in line #230.
4. POKE the location of the trio of bytes for the Cth array variable in locations 8,9 (low,high). This is done in line #240.
5. CALL 768 to start the assembly language search routine. When control returns to Applesoft the array number that has satisfied the search will be returned in locations 30,31. If PEEK(30) + 256*PEEK(31) is greater than N, then the search has failed. If not, then a match has been made with R$(C) where C=PEEK(30) + 256*PEEK(31) and R$ is the array that is being searched.
6. To continue the search to the end of the array, increment C and repeat the above process.

The routine, as written, does not search for exact matches with the string array variables. If the leftmost part of a string array variable is the same as the search phrase, a match is considered to have occurred.

A useful application of this search routine is to use it in conjunction with a mailing list database program. In this way, the search time for an individual record can be cut down dramatically.

Gary B. Little first became interested in computers by writing data analysis programs in FORTRAN on an IBM 370/168 for an M.S. degree in Physical Chemistry (Microwave Spectroscopy). Ultimately he became interested in microcomputing and purchased an Apple II micro two years ago. He is past president of Apple's British Columbia Computer Society, an Apple user group located in Vancouver, B.C. He is currently the treasurer of this group.

MICRO

MICRO Club Circuit

Mike Rowe
Club Circuit
P.O. Box 6502
Chelmsford, MA 01824

The following club announcements are presented in zip code order.

APPLESHARE
The purpose of this club is to allow persons interested in and using Apple Computers to exchange views, problems and experiences. Approximate number of members is 55. Meetings held the last Tuesday of each month (7:30 p.m.), except December, at Computerworks. For more information, contact:
Mr. Jack Adinolfi, President
c/o Computerworks
1439 Post Road East
Westport, Connecticut 06880

Philadelphia Area Computer Society
Subgroups for Apple and PET. Meets the third Saturday of each month at LaSalle College Science Building, 20th at Olney Avenue, Philadelphia, PA. HOTLINE telephone number for meeting news is (215)925-5264. Approximate number of members is 300. For more information, contact:
Eric Hafler, President
P.O. Box 1954
Philadelphia, Pennsylvania
19105

The MicroComputer Investors Assoc.
This group publishes "The MicroComputer Investor". Its purpose is utilizing microcomputers to assist in making and managing investments. General membership meeting will be held at PC81-Philadelphia, PA. For more information, contact:
Jack Williams
902 Anderson Drive
Fredericksburg, Virginia 22401

MICRO offers a free one year subscription to all clubs registered with us. For registration form write to:
MICRO Club Circuit
Box 6502
Chelmsford, MA 01824

A.C.E.S.
The Apple Computer Enjoyment Society publishes a bi-monthly newsletter of approximately 30-40 pages. Their purpose is to inform and coordinate the knowledge of their members about the Apple in the least painful way. Meetings are held at 7:30 p.m. at the N.E. High School, Ft. Lauderdale, FL, on the 1st Wednesday of the month for even months and the 1st Thursday of the month for odd months. Membership is currently at 130+. For more information contact:
Don Lehmbeck, President
P.O. Box 9222
Coral Springs, Florida 33065
(305)524-ACES

Attention Educators
Affiliated with the Cleveland Digital Group, this club's primary objective will be the investigation, discovery, and exchange of functional and innovative Computer-Aided Instruction ideas among interested computer, minicomputer, or microcomputer users and/or owners. Monthly meetings will be held every third Sunday at the Cleveland Heights-University Heights main library, 2345 Lee Road, Cleveland Heights, Ohio. If you're interested, send a self-addressed stamped business envelope to:
Joyce Townsend
P.O. Box 18431
Cleveland Heights, Ohio 44118
or call (216)932-6799

Michigan APPLE
The purpose of this club is to help members have fun with their APPLES by sharing knowledge and information. Publishes a newsletter called "The Michigan APPLE-gram", which is printed 10 times per year. Meetings are held at 7:30 p.m. on the last Tuesday of each month at Southfield-Lathrup High School (12 miles east of Evergreen). Membership is currently at 300. For more information, contact:
Jon Lawrence, President
c/o The Michigan Apple
P.O. Box 551
Madison Heights, Michigan
48071

Dental Computer Newsletter
For Medical & Dental Professionals using Micro & Mini computers for treatment and office purposes. Membership is 1,500+. Meetings are held at address noted below. For more information, contact:
E.J. Neiburger, DDS, President
1000 North Avenue
Waukegan, Illinois 60085

OSIG Chicagoland
Meets the second Monday of each month at 7:30 p.m. to exchange ideas, discoveries, gripes and puzzles. Don Peterson is the President. For more information, contact:
Paul Rainey
c/o York High School
Math Department
355 W. St. Charles Road
Elmhurst, Illinois 60126
(312)530-1240, ext. 256

River City Apple Corps
Meets the 2nd Thursday of each month at 6:30 p.m. at the Old Quarry Library. Approximate number of members is 100. They have a monthly publication called "APPLE-DILLO". For more information, contact:
Lenard Fein
2015 Ford Street
Austin, Texas 78704

APPLEquerque Computer Club
Our purpose is to foster knowledge and use of the Apple computer; to educate our members on successful programming skills and to provide a forum to exchange programs, ideas and techniques. Membership is currently at 38. Meets on the first Tuesday and third Wednesday at a local computer store. For more information, contact:
James (Chuck) Segrest, Pres.
6609 Orphelia Ave. N.E.
Albuquerque, New Mexico
87109

Okinawa Computer Club
Holds their meetings on the second and fourth Sunday of each month at 6:30 p.m. at the Halstead Pavilion (USAF Clinic), Kadena AFB, Okinawa, Japan. For more information contact:
Ralph Tullo
PSC #2 Box 12558
APO SF, California 96367

Alice Apple Users Group
This group meets every fourth Tuesday to exchange S/W and to propose usage of the Apple. Membership is currently at 20. For more information, contact:
Al DeSalvio, President
22 Hablett Crescent
Alice Springs, N.T.
Australia 5750

6502-Club Copenhagen
Address change for this club:
John Svensson
c/o Torvelaengerne, 10
DK—2620 Albertslund
Denmark

The Newest In

Apple Fun

We've taken five of our most popular programs and combined them into one tremendous package full of fun and excitement. This disk-based package now offers you these great games:

Mimic—How good is your memory? Here's a chance to find out! Your Apple will display a sequence of figures on a 3 × 3 grid. You must respond with the exact same sequence, *within the time limit*.

There are five different, increasingly difficult versions of the game, including one that will keep going indefinitely. Mimic is exciting, fast paced and challenging—fun for all!

Air Flight Simulation—Your mission: Take off and land your aircraft without crashing. You're flying blind —on instruments only.

A full tank of fuel gives you a maximum range of about 50 miles. The computer will constantly display updates of your air speed, compass heading and altitude. Your most important instrument is the Angle of Ascent/Bank Indicator. It tells if the plane is climbing or descending, whether banking into a right or left turn.

After you've acquired a few hours of flying time, you can try flying a course against a map or doing aerobatic maneuvers. Get a little more flight time under your belt, the sky's the limit.

Colormaster—Test your powers of deduction as you try to guess the secret color code in this Mastermind-type game. There are two levels of difficulty, and three options of play to vary your games. Not only can you guess the computer's color code, but it will guess yours! It can also serve as referee in a game between two human opponents. Can you make and break the color code...?

Star Ship Attack—Your mission is to protect our orbiting food station satellites from destruction by an enemy star ship. You must capture, destroy or drive off the attacking ship. If you fail, our planet is doomed...

Trilogy—This contest has its origins in the simple game of tic-tac-toe. The object of the game is to place three of your colors, in a row, into the delta-like, multi-level display. The rows may be horizontal, vertical, diagonal and wrapped around, through the "third dimension". Your Apple will be trying to do the same. You can even have your Apple play against itself!

Minimum system requirements are an Apple II or Apple II Plus computer with 32K of memory and one minidisk drive. Mimic requires Applesoft in ROM, all others run in RAM or ROM Applesoft.

Order No. 0161AD $19.95

Solar Energy For The Home

With the price of fossil fuels rising astronomically, solar space-heating systems are starting to become very attractive. But is solar heat cost-effective for you? This program can answer that question.

Just input this data for your home: location, size, interior details and amount of window space. It will then calculate your current heat loss and the amount of gain from any south facing windows. Then, enter the data for the contemplated solar heating installation. The program will compute the NET heating gain, the cost of conventional fuels vs. solar heat, and the calculated payback period—showing if the investment will save you money.

Solar Energy for the Home: It's a natural for architects, designers, contractors, homeowners...anyone who wants to tap the limitless energy of our sun.

Minimum system requirements are an Apple II or Apple II Plus with one disk drive and 28K of RAM. Includes AppleDOS 3.2.

Order No. 0235AD (disk-based version) $34.95

Math Fun

The Math Fun package uses the techniques of immediate feedback and positive reinforcement so that students can improve their math skills while playing these games:

Hanging—A little man is walking up the steps to the hangman's noose. But YOU can save him by answering the decimal math problems posed by the computer. Correct answers will move the man down the steps and cheat the hangman.

Spellbinder—You are a magician battling a computerized wizard. In order to cast death clouds, fireballs and other magic spells on him, you must correctly answer problems involving fractions.

Whole Space—Pilot your space craft to attack the enemy planet. Each time you give a correct answer to the whole number problems, you can move your ship or fire. But for every wrong answer, the enemy gets a chance to fire at you.

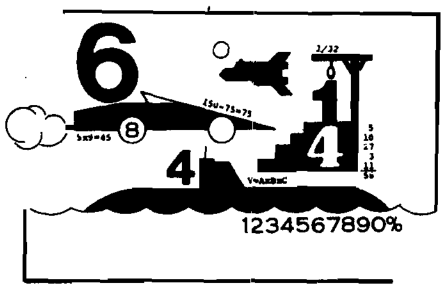
Car Jump—Make your stunt car jump the ramps. Each correct answer will increase the number of buses your car must jump over. These problems involve calculating the areas of different geometric figures.

Robot Duel—Fire your laser at the computer's robot. If you give the correct answer to problems on calculating volumes, your robot can shoot at his opponent. If you give the wrong answer, your shield power will be depleted and the computer's robot can shoot at yours.

Sub Attack—Practice using percentages as you maneuver your sub into the harbor. A correct answer lets you move your sub and fire at the enemy fleet.

All of these programs run in Applesoft BASIC, except Whole Space, which requires Integer BASIC.

Order No. 0160AD $19.95



Paddle Fun

This new Apple disk package requires a steady eye and a quick hand at the game paddles! It includes:

Invaders—You must destroy an invading fleet of 55 flying saucers while dodging the carpet of bombs they drop. Your bomb shelters will help you—for a while. Our version of a well known arcade game! Requires Applesoft in ROM.

Howitzer—This is a one or two person game in which you must fire upon another howitzer position. This program is written in HIGH-RESOLUTION graphics using different terrain and wind conditions each round to make this a demanding game. The difficulty level can be altered to suit the ability of the players. Requires Applesoft in ROM.

Space Wars—This program has three parts: (1) Two flying saucers meet in laser combat—for two players, (2) two saucers compete to see which can shoot out the most stars—for two players, and (3) one saucer shoots the stars in order to get a higher rank—for one player only. Requires Applesoft.

Golf—Whether you win or lose, you're bound to have fun on our 18 hole Apple golf course. Choose your club and your direction and hope to avoid the sandtraps. Losing too many strokes in the water hazards? You can always increase your handicap. Get off the tee and onto the green with Apple Golf. Requires Applesoft.

The minimum system requirement for this package is an Apple II or Apple II Plus computer with 32K of memory and one minidisk drive.

Order No. 0163AD $19.95

Skybombers

Two nations, seperated by The Big Green Mountain, are in mortal combat! Because of the terrain, their's is an aerial war—a war of SKYBOMBERS!

In this two-player game, you and your opponent command opposing fleets of fighter-bombers armed with bombs and missiles. Your orders? Fly over the mountain and bomb the enemy blockhouse into dust!

Flying a bombing mission over that innocent looking mountain is no milk run. The opposition's aircraft can fire missiles at you or you may even be destroyed by the bombs as they drop. Desperate pilots may even ram your plane or plunge into your blockhouse, suicidally.

Flight personnel are sometimes forced to parachute from badly damaged aircraft. As they float helplessly to earth, they become targets for enemy missiles.

The greater the damage you deal to your enemy, the higher your score, which is constantly updated at the bottom of the display screen.

The sounds of battle, from exploding bombs to the pathetic screams from wounded parachutists, remind each micro-commander of his bounden duty. Press On, SKYBOMBERS—Press On!

Minimum system requirements: An Apple II or Apple II Plus, with 32K RAM, one disk drive and game paddles.

Order No. 0271AD (disk-based version) $19.95

Instant Software™

*A trademark of Apple Computer Inc.

PETERBOROUGH, N.H. 03458
603-924-7296

Apple* Software

From Instant Software

Santa Paravia and Fiumaccio

Buon giorno, signore!

Welcome to the province of Santa Paravia. As your steward, I hope you will enjoy your reign here. I feel sure that you will find it, shall we say, profitable.

Perhaps I should acquaint you with our little domain. It is not a wealthy area, signore, but riches and glory are possible for one who is aware of political realities. These realities include your serfs. They constantly request more food from your grain reserves, grain that could be sold instead for gold florins. And should your justice become a trifle harsh, they will flee to other lands.

Yet another concern is the weather. If it is good, so is the harvest. But the rats may eat much of our surplus and we have had years of drought when famine threatened our population.

Certainly, the administration of a growing city-state will require tax revenues. And where better to gather such funds than the local marketplaces and mills? You may find it necessary to increase custom duties or tax the incomes of the merchants and nobles. Whatever you do, there will be far-reaching consequences. . .and, perhaps, an elevation of your noble title.

Your standing will surely be enhanced by building a new palace or a magnificent *cattedrale*. You will do well to increase your landholdings, if you also equip a few units of soldiers. There is, alas, no small need for soldiery here, for the unscrupulous Baron Peppone may invade you at any time.

To measure your progress, the official cartographer will draw you a *mappa*. From it, you can see how much land you hold. how much of it is under the plow and how adequate your defenses are. We are unique in that here, the map IS the territory.

I trust that I have been of help, signore. I look forward to the day when I may address you as His Royal Highness, King of Santa Paravia. *Buona fortuna* or, as you say, "Good luck". For the Apple 48K.

Order No. 0174A $9.95 (cassette version).

Order No. 0229AD $19.95 (disk version).

TO ORDER SEE YOUR LOCAL INSTANT SOFTWARE DEALER OR USE THE ORDER FORM BELOW

For Fast Service *call now* Toll-Free 1-800-258-5473

Apple Cassettes

0018A Golf.............................. $7.95
0025A Mimic............................ $7.95
0040A Bowling/Trilogy.................... $7.95
0073A Math Tutor I....................... $7.95
0079A Oil Tycoon........................ $9.95
0080A Sahara Warriors.................... $7.95
0088A Accounting Assistant................ $7.95
0094A Mortgage w/Prepayment Option/ Financier.......................... $7.95
0096A Space Wars........................ $7.95
0098A Math Tutor II...................... $7.95
0174A Santa Paravia and Fiumaccio.......... $9.95
0148A Air Flight Simulation................ $9.95

We Guarantee It!

Instant Software Guarantee

OUR PROGRAMS ARE GUARANTEED TO BE QUALITY PRODUCTS. IF NOT COMPLETELY SATISFIED YOU MAY RETURN THE PROGRAM WITHIN 60 DAYS. A CREDIT OR REPLACEMENT WILL BE WILLINGLY GIVEN FOR ANY REASON.

109

Name ______

Address ______

City ______ State ______ Zip ______

☐ Check ☐ Money Order ☐ VISA ☐ AMEX ☐ Master Charge

Card No. ______ Exp. Date ______

Signed ______ Date ______

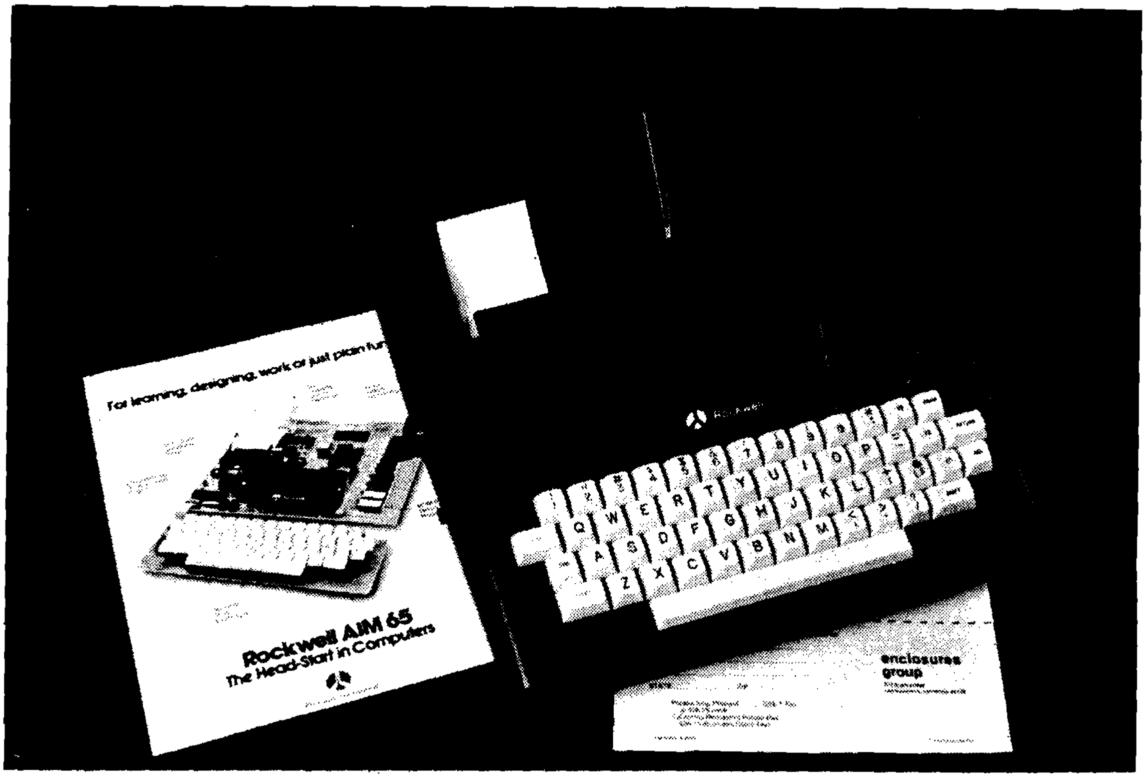
Order your Instant Software today!

Quantity	Order No.	Program name	Unit cost	Total cost
		Shipping and handling		$1.00
			Total order	

Instant Software Inc.™

Peterborough, N.H. 03458

PERFECT AIM



ATTRACTIVE FUNCTIONAL PACKAGING FOR YOUR AIM-65 MICROCOMPUTER

- Professional Appearance
- Striking Grey and Black Color Combination
- Protects Vital Components

ENGINEERED SPECIFICALLY FOR THE ROCKWELL AIM-65

- All Switches Accessible
- Integral Reset Button Actuator
- Easy Paper Tape Replacement

EASILY ASSEMBLED

- Absolutely No Alteration of AIM-65 Required
- All Fasteners Provided
- Goes Together in Minutes

MADE OF HIGH IMPACT STRENGTH THERMOFORMED PLASTIC

- Kydex 100*
- Durable
- Molded-In Color
- Non-Conductive

AVAILABLE FROM STOCK

- Allow Three to Four Weeks for Processing and Delivery
- No COD's Please
- Dealer Inquiries Invited

TO ORDER: 1. Fill in this Coupon (Print or Type Please)
2. Attach Check or Money Order and Mail to:

NAME ____________________

STREET ____________________

CITY ____________________

STATE ______________ ZIP ________

SAE 1-1 PLEASE SHIP PREPAID ________ SAE 1-1(s)
@$43.50 each
California Residents Please Pay
$46.33 (Includes Sales Tax)

SAE 1-2 PLEASE SHIP PREPAID ________ SAE 1-2(s)
@$46.50 each
California Residents Please Pay
$49.52 (Includes Sales Tax)

enclosures group

786 bush street
san francisco, california 94108

***TM Rohm & Hass** **Patent Applied For**

Interfacing the 6522 Versatile Interface Adapter

The 6522 handles I/O for AIM, SYM, KIM, PET and Ohio Scientific microcomputer systems. This chip contains latches, counters, timers, shift registers, and handshaking lines, and provides a great deal of support for many interfacing applications. Here is how to include these capabilities in your system.

Marvin L. De Jong
Math/Physics Department
The School of the Ozarks
Pt. Lookout, Missouri 65726

One of the most popular interfacing integrated circuits in the 6500 family is the 6522 Versatile Interface Adapter. It is also a complex chip, and becoming familiar with it can be frustrating. In an earlier article published in MICRO, October 1979, "6522 Timing and Counting Techniques" (17:27) we described how to use the T1 timer and the T2 counter/timer. In this article we give more of an overview of the entire chip, we supply some detailed interfacing circuits for adding 6522s to your microcomputer system, and we provide some useful charts that diagram the functions of the bits in various registers. I have found these latter charts to be very useful when writing assembly language programs that use the 6522 as an I/O device. With regard to the interfacing circuits included in this article, note that several 6502 systems manufacturers are offering prototyping boards that mate with the motherboards or card file systems they sell. The interfacing circuits given herein are appropriate for wiring your own I/O board that might include several 6522s as well as other integrated circuits such as A/D converters, V/F converters, and D/A converters.

Several single-board microcomputers use a 6522 as *the* I/O device. For example, the AIM 65, SYM-1, and the SUPERKIM make extensive use of the 6522 either as an on-board I/O device, or for the user's applications.

Interfacing the 6522

The 6522 is a 40-pin integrated circuit that provides the user with two eight-bit I/O ports (each with two handshaking pins), a serial data port, and two 16-bit timer/counters. A pin configuration diagram is given in figure 1, and a block diagram is shown in figure 2. The 6522 has 16 internal registers that are selected by the logic levels on the four register select lines RS3 - RS0. Table 1 summarizes the names of the various registers that are selected by RS3 - RSO and the R/$\overline{W}$ line. The I/O ports are sometimes referred to as PAD and PBD, rather than the designations given in table 1. All of the functions of the 6522 are initiated and controlled by reading or writing to the registers listed in table 1. Thus, our first task is to connect the 6522 to a 6500 family microprocessor, the 6502 for example, so that we can read and write to the registers.

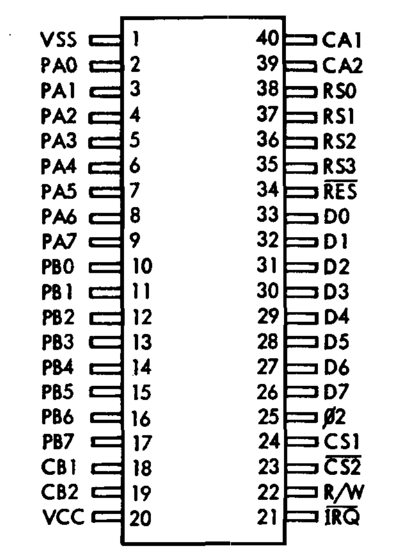


***Figure 1.* Pin configuration of the 6522. Courtesy of Rockwell International Corporation.**

Refer to the block diagram of the 6522 shown in figure 2. The pins that concern us in the present context are the so-called "chip access control" pins, the $\overline{IRQ}$ pin, and the pins D7 - D0 that connect to the microcomputer system's data bus. We begin with the chip access control pins and the $\overline{IRQ}$ pin. Several of these connections are straightforward: $\overline{RES}$ will be connected to the system reset line and the $\overline{RES}$ pin on the microprocessor, R/$\overline{W}$ is connected to the R/$\overline{W}$ line on the microprocessor, 5_2 goes to the microprocessor's 5_2, and the $\overline{IRQ}$ pin is connected to the $\overline{IRQ}$ pin on the 6502. (A pull-up resistor, nominally 4.7K ohms, is required for this latter connection.)

The remaining chip access control pins are used to address the 16 internal registers of the 6522. If pins RS3 - RSO are connected to address lines A3 - A0, respectively, then the 16 registers will occupy 16 contiguous memory locations in the address space of the computer system. The registers will have the same order given in table 1. The location of the block of 16 memory locations in the address space of the computer system will be determined by how the high-order address lines are decoded to form the device-select pulses that are applied to the chip-select pins CS1 and $\overline{CS2}$.

The design philosophy of most 6502 systems calls for R/W memory to be located at the low end of the address space and the microcomputer system ROM is located at the high end of the address space. Thus, I/O functions are usually located somewhere in the "middle" of the address space. An address decoding scheme to do this is described in figures 3 and 4.

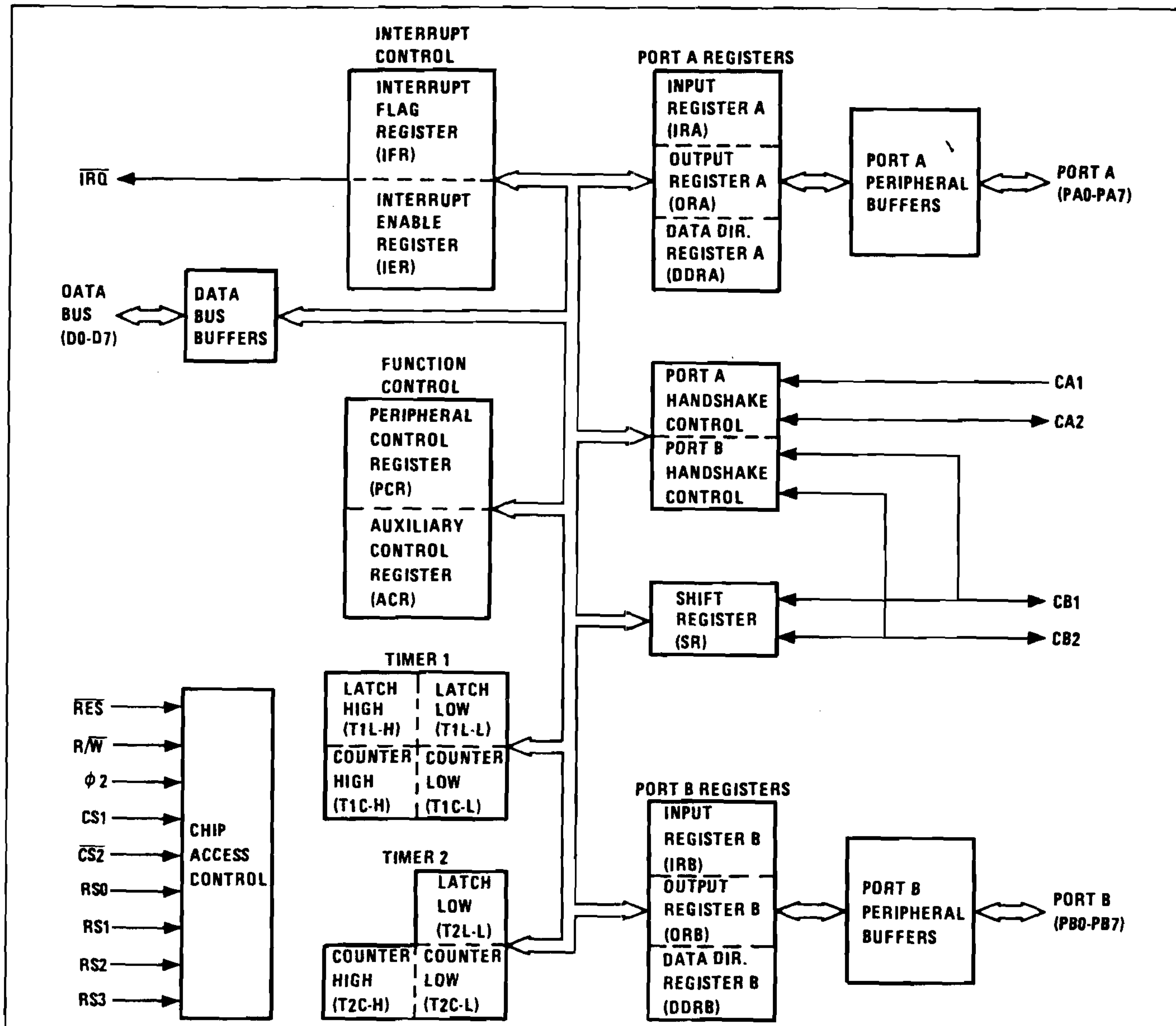


***Figure 2.* Block diagram of the 6522. Courtesy of Rockwell International Corporation.**

Assume that a page of memory space is to be dedicated to I/O functions, giving 256 locations. The 74LS04 and the 74LS30 in figure 3 are used to select any page of memory that ends in a three, seven, 11, or 15 such as pages \$03, \$67, \$93, \$AB, or \$FF. The jumpers or switches connected to the 74LS30 select the page of memory that is to be used for I/O. This decoding scheme should allow the user sufficient flexibility to avoid both the R/W memory locations and the ROM locations.

The page select pulse, $\overline{\text{DS\$93XX}}$ in figure 3, is connected to a 74LS154 one-of-sixteen decoder that decodes address lines A7 - A4. This decoder determines the high-order nibble of the low-order byte of the address in the page selected by the circuit of figure 3.

Register Number	RS Coding RS3	RS2	RS1	RS0	Register Desig.	Description Write	Read
0	0	0	0	0	ORB/IRB	Output Register "B"	Input Register "B"
1	0	0	0	1	ORA/IRA	Output Register "A"	Input Register "A"
2	0	0	1	0	DDRB	Data Direction Register "B"	
3	0	0	1	1	DDRA	Data Direction Register "A"	
4	0	1	0	0	T1C-L	T1 Low-Order Latches	T1 Low-Order Counter
5	0	1	0	1	T1C-H	T1 High-Order Counter	
6	0	1	1	0	T1L-L	T1 Low-Order Latches	
7	0	1	1	1	T1L-H	T1 High-Order Latches	
8	1	0	0	0	T2C-L	T2 Low-Order Latches	T2 Low-Order Counter
9	1	0	0	1	T2C-H	T2 High-Order Counter	
10	1	0	1	0	SR	Shift Register	
11	1	0	1	1	ACR	Auxiliary Control Register	
12	1	1	0	0	PCR	Peripheral Control Register	
13	1	1	0	1	IFR	Interrupt Flag Register	
14	1	1	1	0	IER	Interrupt Enable Register	
15	1	1	1	1	ORA/IRA	Same as Reg 1 Except No "Handshake"	

***Table 1.* Addressing the 16 registers of the 6522. Courtesy of Synertek, Inc.**

The 74LS154 circuit is given in figure 4. It devides a page of memory into blocks of 16 memory locations. Although only one of the device select pulses, $\overline{DS\$930X}$, is shown connected to an I/O device, the other device select pulses may be used to select other devices as they are added to your I/O board.

The 6522 decodes the address lines A3 - A0 and addresses its internal registers when $\overline{CS2}$ is at logic zero *and* CS1 is at logic one. Thus, in figure 4 the device select pulse from the 74LS154 is connected to $\overline{CS2}$, while CS1 is held at logic one. The 16 registers on the 6522 have addresses $9300 to $930F starting with the PBD register at $9300 and ending with the PAD "No Handshake" register at location $930F. Another 6522 could be added using any of the other 15 device selects from the 74LS154.

As mentioned before, the 74LS154 in figure 4 divides a page of memory into blocks of 16 locations. Other 74LS154s can be added to divide each of these blocks into separate memory locations; that is, device select pulses can be produced for each memory location in a block of 16. The circuit of figure 5 indicates how this is accomplished. Although each 6522 added to the system requires 16 locations, there should be enough locations left, in the page that we choose for I/O functions, to handle our requirements.

Suppose we wish to mount all the I/O circuits on a single I/O board (or card) to be mounted on some kind of motherboard. The circuits in figures 3 to 5 could easily be wired on such a card with several 6522 VIAs and other I/O devices. In such a case, buffering of the data bus is usually implemented on the board, and frequently the address bus and the control bus are also buffered. The page select pulse from the SN74LS30 in figure 3 can be used to activate the data bus buffers when an I/O device on the card is selected. The circuit in figure 6 indicates one possible way to buffer the data bus using two SN74LS243 quadruple bus transceivers for buffers.

In figure 6 only one of the SN74LS243 buffers is shown in detail. Note that the buffers, three-state devices, are activated from left to right with a WRITE operation; that is, when the page select pulse is at logic zero and the R/$\overline{W}$ line is at logic zero. The buffers are activated from right to left with a READ operation; that is, when the

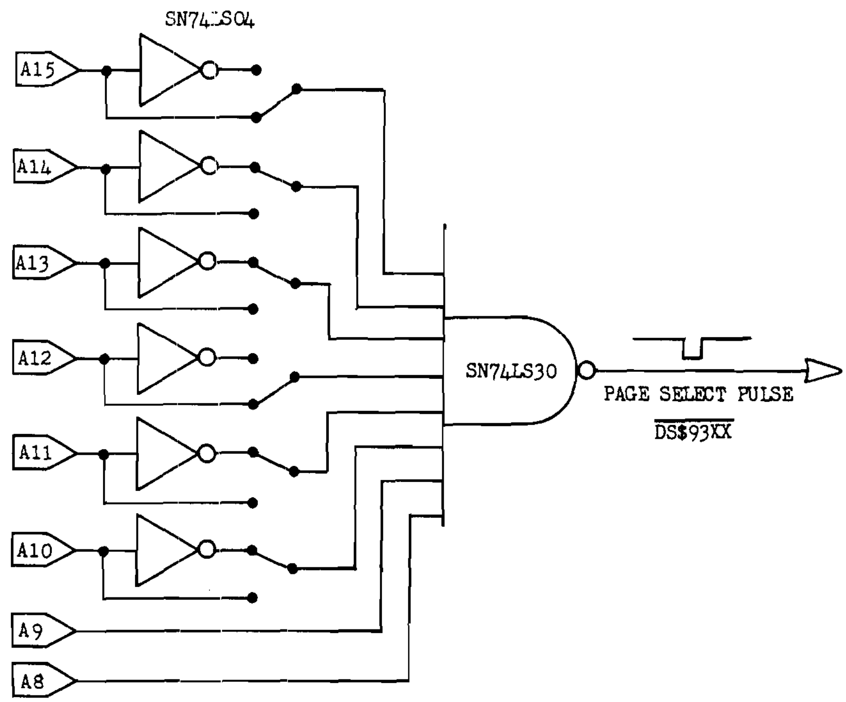


Figure 3. **Programmable page select circuit, preset for page $93 of the address space.**

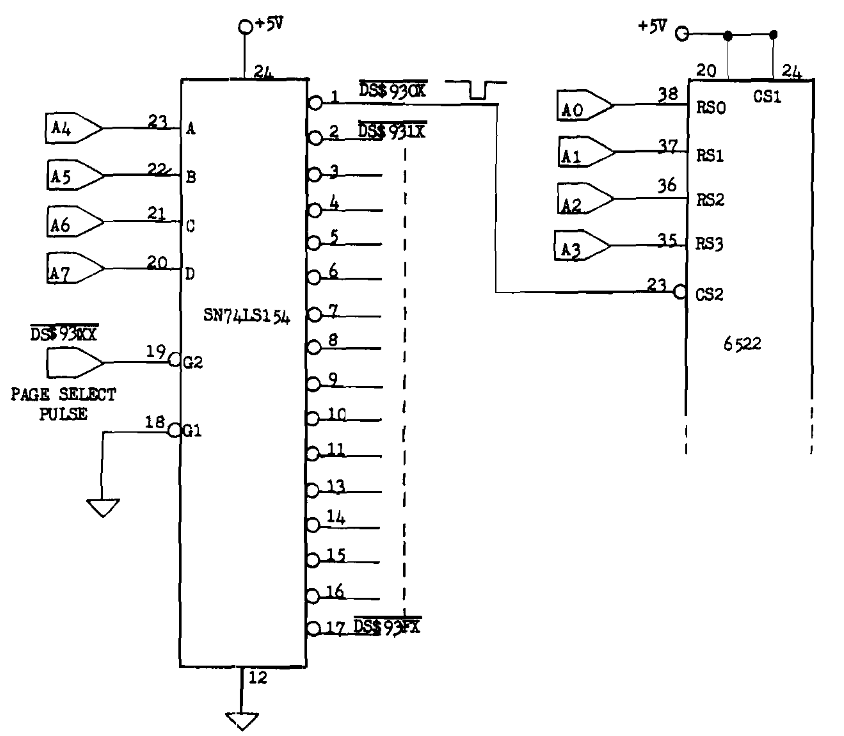


Figure 4. **Using a SN74LS154 to select a 6522. The 6522 occupies 16 memory locations with addresses $9300 to $930F. The page select signal is generated by the circuit in Figure 3.**

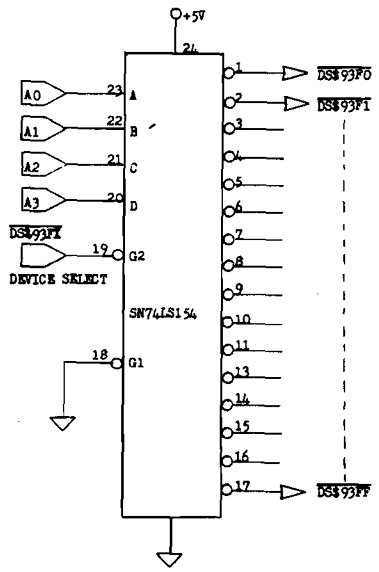


***Figure* 5. Using a 74LS154 to provide device select pulses for memory locations $93F0 - $93FF. The device select pulse DS$93FX is from the 74LS154 in Figure 4.**

page select pulse is at logic zero and the R/W line is at logic one.

The 6522 VIA's data pins are connected to the DB7 - DB0 pins on the right-hand side of the SN74LS243s. Any other I/O devices will also be using these data lines. If the only I/O device on the I/O board is a single 6522, then data bus buffering might not be required.

Figure 7 illustrates how the address bus and the control bus may be buffered. The three-state devices shown are kept in their active states at all times. The 81LS97 is convenient to use because it buffers eight lines with one chip, so only two chips are required to buffer the entire address bus. Many other chips will do the same job, and you should feel free to substitute these.

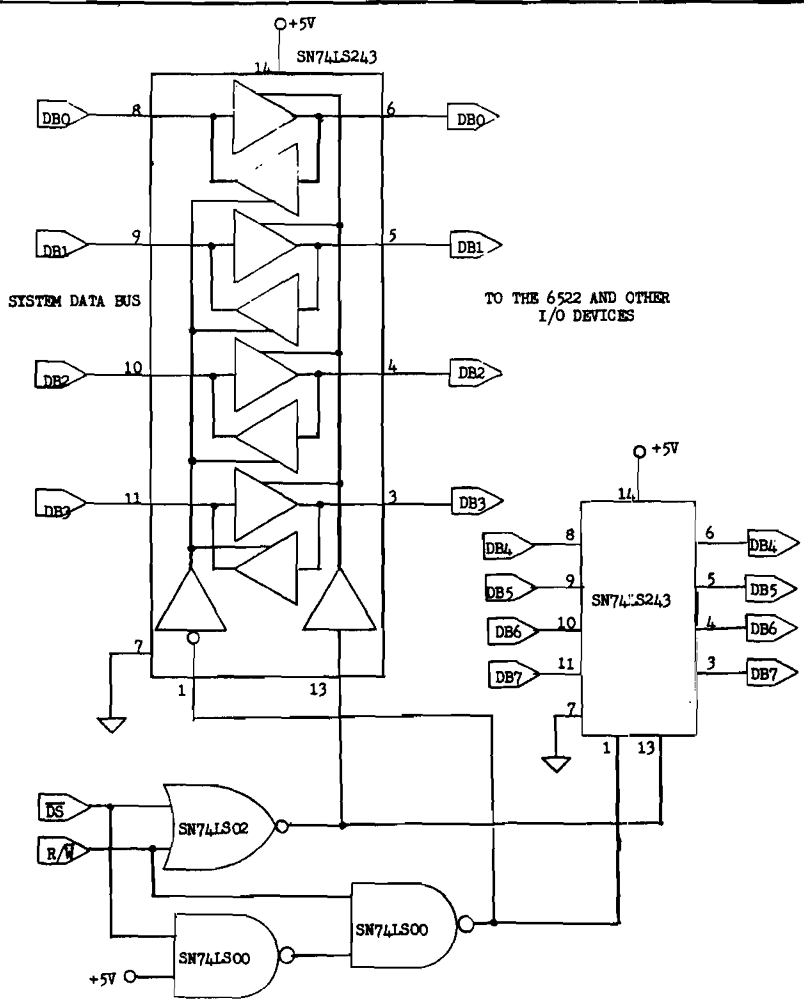


***Figure* 6. Buffering the data bus for the I/O devices. DS is the page select signal from the 74LS30 in Figure 3. R/W is the system READ/WRITE line.**

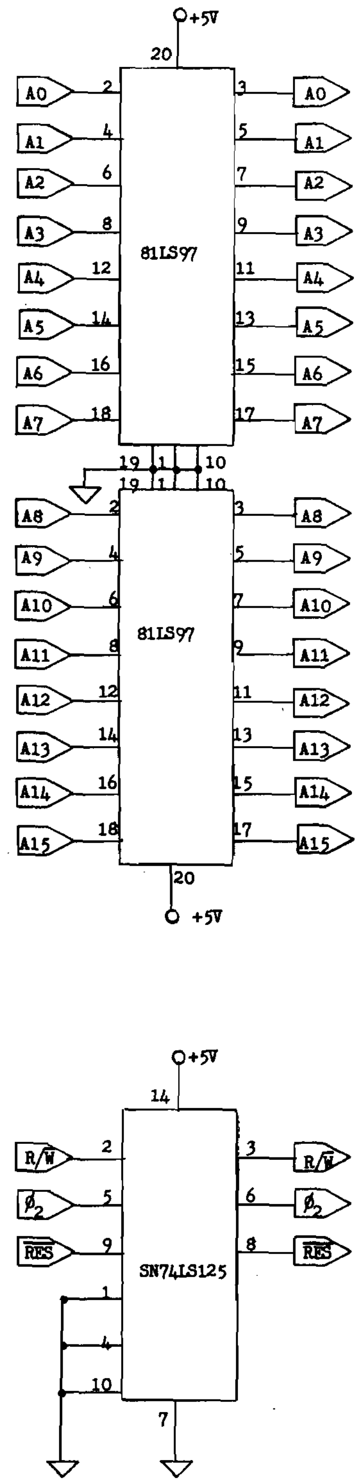


***Figure* 7. Buffering the address bus and the control bus.**

Example 1. A program to make Port A an input port and Port B an output port.

```
$0300 A9 FF    START   LDA $FF   Store all ones in the data direction
$0302 8D 02 93         STA PBDD  register of Port B.
$0305 A9 00            LDA $00   Store all zeros in the data direction
$0307 8D 03 93         STA PADD  register of Port A.
```

Table 2. Memory locations of the 16 6522 registers.

ADDRESS	REGISTER
$9300	Port B — PBD
$9301	Port A — PAD
$9302	Port B Data Direction — PBDD
$9303	Port A Data Direction — PADD
$9304	Timer 1 Latch Low and Timer 1 Counter Low — T1LL T1CL
$9305	Timer 1 Latch High and Timer 1 Counter High — T1LH & T1CH
$9306	Timer 1 Latch Low — T1LL
$9307	Timer 1 Latch High — T1LH
$9308	Timer 2 Latch Low and Timer 2 Counter Low — T2LL & T2CL
$9309	Timer 2 Counter High — T2CH
$930A	Shift Register — SR
$930B	Auxiliary Control Register — ACR
$930C	Peripheral Control Register — PCR
$930D	Interrupt Flag Register — IFR
$930E	Interrupt Enable Register — IER
$930F	Port A — PAD (Without handshaking)

Simple Input/Output Functions with the 6522

Now that we have interfaced the 6522 to a 6502 (or other 6500 family microprocessor), we are ready to interface it to some devices in the world outside of the microprocessor system.

There is a one-to-one correspondence between the pin functions (input or output) of a port and the bit values in the port's corresponding data direction register. A one in the data direction register bit configures the corresponding I/O pin to be an output pin, while a zero in the data direction register bit configures the corresponding I/O pin to be an input pin. Thus, if $F8 is loaded into PBDD, address $9302, using the decoding illustrated in the figures, then pins 7, 6, 5, 4, and 3 of Port B (PBD) will be output pins, while pins 2, 1, and 0 will be input pins. Example 1 illustrates how all eight pins of Port B are configured as output pins, while all eight pins of Port A are configured as input pins.

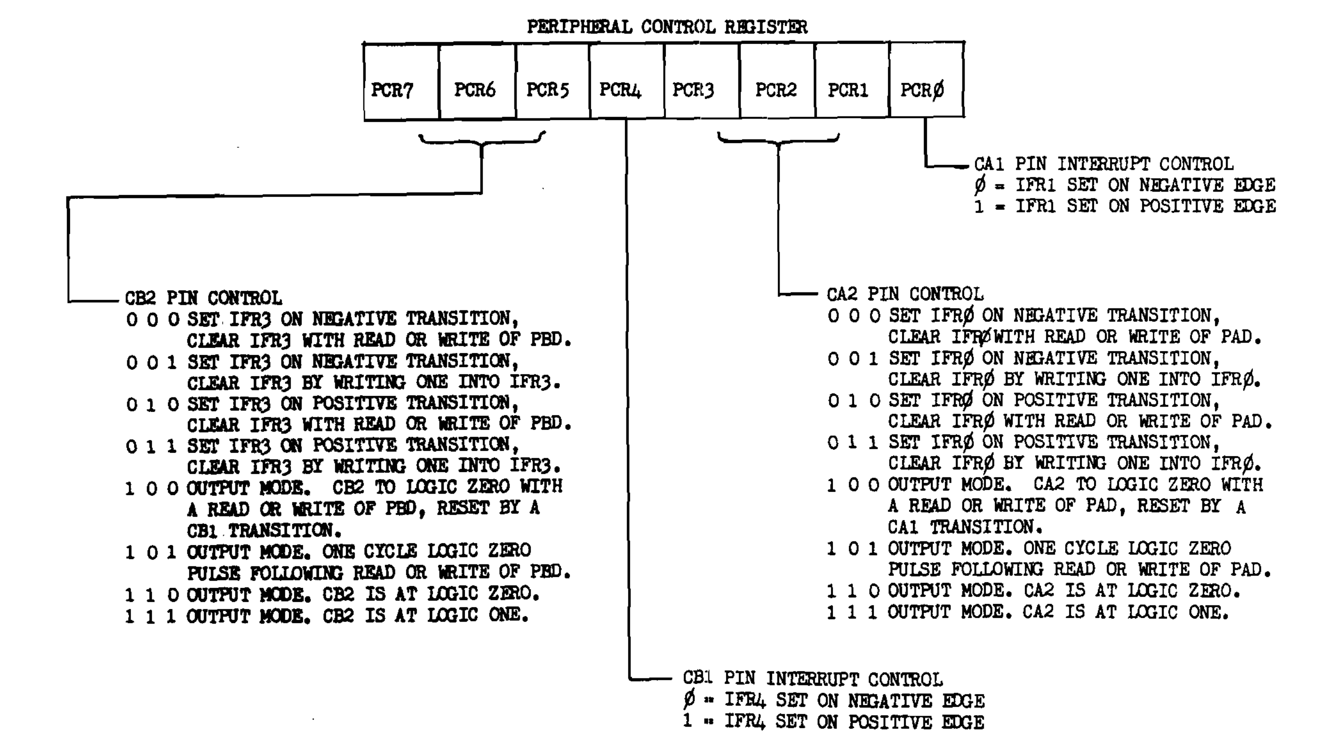


Figure 8. **Determining the function of the control pins CA1, CA2, CB1, and CB2 by the number stored in the peripheral control register (PCR).**

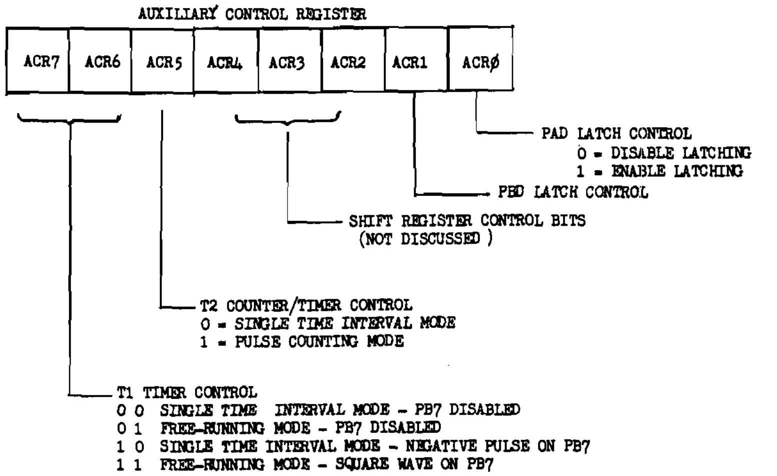


Figure 9. **Controlling various 6522 functions with the auxiliary control register (ACR).**

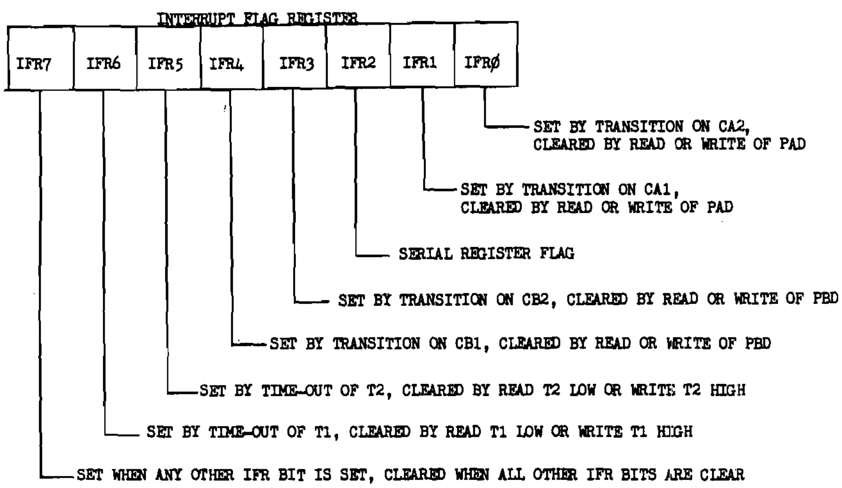


Figure 10. **The interrupt flag register (IFR). Any bit except IFR7 may also be cleared by writing a one into that bit.**

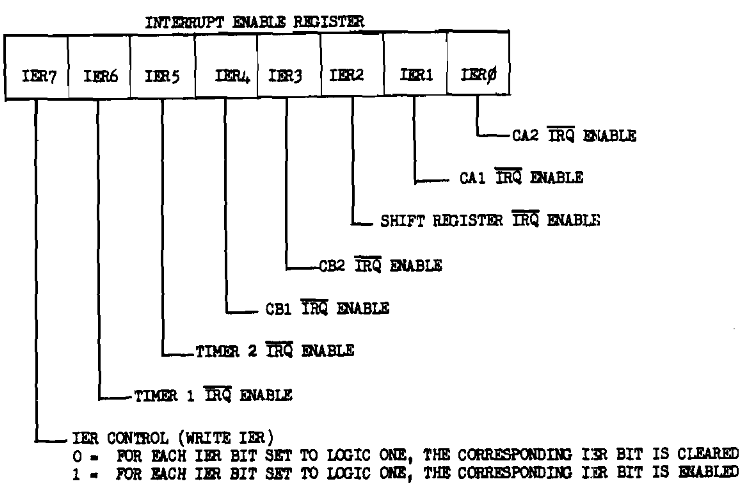


Figure 11. **Controlling the $\overline{IRQ}$ pin on the 6522. If the $\overline{IRQ}$ pin is enabled by a bit in the IER, then the 6522 $\overline{IRQ}$ pin will go to logic zero when the corresponding flag in the interrupt flag register is set.**

Once the I/O port has been configured, data may be written to (STA instruction) an output port and read at (LDA instruction) an input port.

It is important to know that when the microcomputer is reset (a condition that also occurs during power-up conditions) all of the registers except the timer latches are cleared. Thus, the Port A and Port B pins are in the input condition after a reset. TTL integrated circuits whose inputs may be connected to the Port A or Port B pins will behave as if their inputs are at logic one. For example, if one of the pins of Port A controls a TTL circuit that in turn controls a motor, it is important that a logic one at that pin corresponds to the motor being in its off condition, because otherwise, the motor will start running as soon as power is applied to the computer.

It is also important to know that reading Port A gives the values of the logic levels on the Port A *pins*. Reading Port B, on the other hand, gives the logic levels stored in the *output register*, regardless of whether the pins are being loaded or not. Care must be used in using the read-modify-write instructions (ASL, LSR, INC, etc.) to operate on Port A, whereas these instructions will work for Port B.

The Functions of the Other 6522 Registers

Although space does not permit an exhaustive explanation of all of the I/O options, we pause here to briefly describe the purpose of several of the control and flag registers on the 6522. Table 2 summarizes the addresses of all of the registers if the address decoding scheme described in figures 3 to 5 is used. In this section the peripheral control register (PCR), the auxiliary control register (ACR), the interrupt flag register (IFR), and the interrupt enable register (IER) are introduced. In the previous section the two I/O registers, PAD and PBD, were introduced, as well as their associated data direction registers, PADD and PBDD.

The peripheral control register controls the functions of the control pins on the 6522, namely CA1, CA2, CB1, and CB2. The significance of the various bit values in this register is outlined in figure 8. The auxiliary control register controls the modes of the two counter/timers, the shift register, and controls the latching function of

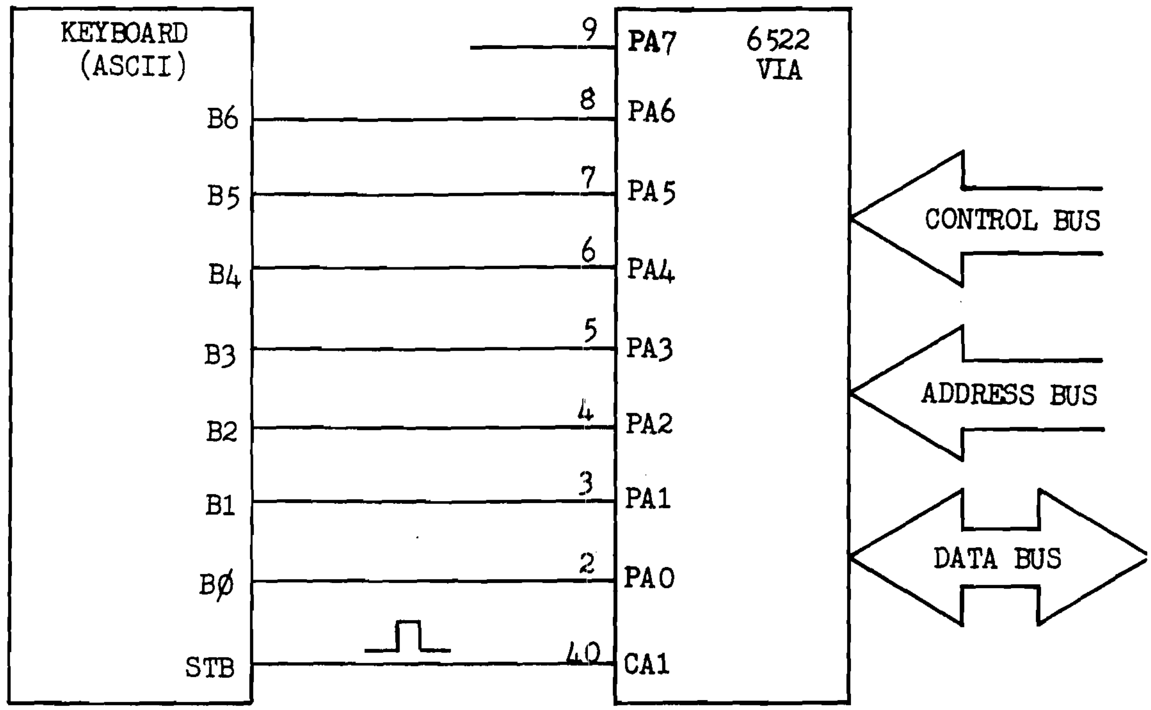


***Figure 12.* Connecting a 6522 VIA to an ASCII encoded keyboard with a strobe pulse.**

Example 2. A program to initialize the 6522 registers to read an ASCII keyboard with strobe.

```
$0300 A9 01     START     LDA $01   Enable latching on Port A with a trans-
$0302 8D 0B 93            STA ACR   ition on CA1 by setting ACR0 to one.
$0305 8D 0C 93            STA PCR   Make CA1 active on positive
  .                        .        transition.
  .                        .        Remainder of main program is here.
  .                        .
$1414 20 00 20            JSR       Jump to subroutine to read keyboard.
                          KEYREAD
  .                        .
  .                        .        More of the main program is here.
  .                        .
$2000 A9 02     KEYREAD   LDA $02   Mask all bits of the IFR except IFR 1.
$0202 2C 0D 93  WAITKEY   BIT IFR   Is IFR1 set?
$0205 F0 FB               BEQ       No. Wait until key is pressed.
                          WAITKEY
$0207 AD 01 93            LDA PAD   Yes. Read keyboard and clear IFR1.
$020A 60                  RTS       Return from subroutine.
```

Example 3. A program to read an ASCII encoded keyboard on an interrupt basis.

```
$0300 A9 01     START      LDA $01   Enable latching on Port A with a trans-
$0302 8D 0B 93             STA ACR   ition on CA1 by setting ACR0 to one.
$0305 8D 0C 93             STA PCR   Make CA1 active on a positive trans-
$0308 A9 82                LDA $82   ition. Bits 7 and 1 are at logic one
$030A 8D 0E 93             STA IER   to enable interrupt from IFR1.
$030D 58                   CLI       Allow interrupts from IRQ.
  .                         .
  .                         .        Remainder of main program is here.
  .                         .
$2000 A9 7F     INTERRUPT  LDA $7F   Mask bit 7 of the input port so the
$2002 2D 01 93             AND PAD   ASCII code is in the accumulator,
$2005 40                   RTI       then return to the main program.
```

the two ports. The significance of the various bit values in this register are outlined in figure 9. Both of these registers are initialized by writing the appropriate binary number to the addresses given in table 2.

On the other hand, the interrupt flag register's contents are usually determined by external events, rather than writing to the register. For example, a negative transition on CA1 may set bit IFR1, causing the program to make a branch. Reading Port A would clear the same bit. (Our example of an ASCII encoded keyboard will make this clear in a few moments.) The various flags in the interrupt flag register are summarized in figure 10.

Finally, the interrupt enable register is used to choose between the option of having an event, say the time-out of the T1 timer, set a flag or set a flag *and* cause an interrupt request ($\overline{IRQ}$ signal). If, for example, IER6 is set by writing $C0 to the interrupt enable register (IER), then the time-out of timer T1 will cause the $\overline{IRQ}$ pin on the 6522 to go to logic zero. If, on the other hand, IER6 is cleared by writing $80 to the interrupt enable register, then the time-out of timer T1 will not produce an $\overline{IRQ}$ signal. (Refer to figure 11 for details of the operation of the IER.)

Now that I have introduced these four registers, let me illustrate their use with an example, namely an input port for an ASCII encoded keyboard. Figure 12 shows the connections that are required. It is assumed that the encoded keyboard provides a positive strobe when the seven bits of ASCII data are available. The program necessary to initialize the PCR and read the keyboard is given in example 2. We will set up the ACR to latch the data from the keyboard into Port A when the strobe occurs, and the keyboard is read by a subroutine that waits until the IFR1 flag is set before reading the keyboard. Such a program might be part of a BASIC interpreter in which the interpreter waits for keyboard entries and then stores them in memory as they are made. Note that the accumulator passes the ASCII code, plus the value of PA7, from the subroutine to the main program where it is processed. This program should be studied in connection with figures 8 to 12.

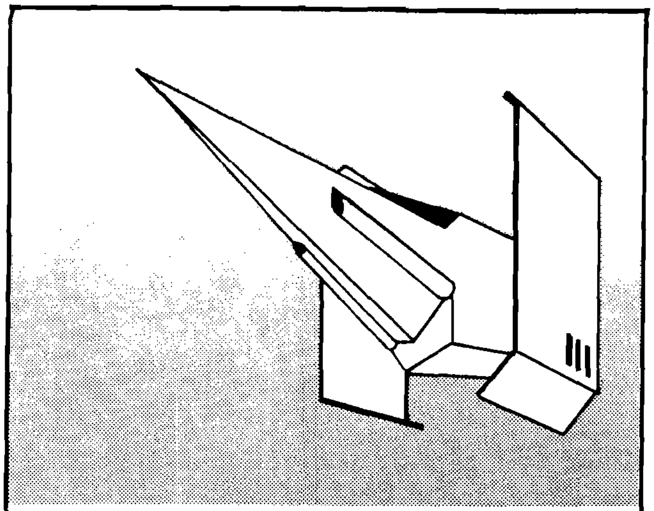
The keyboard could also be read on an interrupt basis. The modifications for this are shown in example 3. Note that the interrupt (IRQ) vector must point to the interrupt routine in order for example 3 to work. Also note that in both example 2 and 3 the Port A data direction register was not initialized to contain $00 because it was assumed that a system power-up or reset accomplished this.

In example 3 there would very likely be more instructions in the interrupt routine to process the keyboard data. For example, it might be stored in some kind of first-in-first-out (FIFO) memory. The important point of example 3 is the means by which the interrupt enable register (IER) is initialized.

Other uses for the control pins will appear in subsequent applications. For example, the control pins in the output mode may be used to initiate a conversion when D/A or A/D converters are interfaced to the 6522. Alternatively, in the input mode the control pins may be used to detect when a conversion is complete. This summarizes our overview of the 6522.

The figures describing the bit functions of the control and flag registers will become a valuable reference for further work with the 6522.

MICRO™



SIRIUS SOFTWARE PRESENTS

Action Software For The Apple

Both Barrels

This package features two games: HIGH NOON and DUCK HUNT. Fun for the very young and the young at heart...you'll love the bad guy that falls off the roof and the dogs fighting over the ducks.

Star Cruiser

STAR CRUISER is a fast action arcade game that can be played by ages 3 and up. SOFTALK magazine rated this one number three...need we say more?

Cyber Strike

CYBER STRIKE This is brand new game for the APPLE II...a hi-res action adventure in space with a full 48k of Assembly Language programming with animation and 3-D effects you haven't seen before. MIND BOGGLING! Everyone said a game like this wasn't possible on the APPLE II, but we did it. Also includes a real time clock (software implemented) and several levels of play. WARNING...THIS GAME REQUIRES PRACTICE TO PLAY SUCCESSFULLY! Uses either 13 or 16 sector APPLE II, II+, or III.

Sirius Software

1537 Howe Ave., Suite 106, Sacramento, CA 95825

HAS YOUR APPLE READ ANY GOOD PROGRAMS LATELY?

APPLE II DISK SOFTWARE

DATA BASE MANAGER
IFO PROGRAM

The IFO (INFORMATION FILE ORGANIZER) can be used for many applications such as: Sales Activity, Check Registers, Balance Sheets, Client/Patient Records, Laboratory Data Reduction, Prescription Information, Grade Records, Mailing Lists, A/R, Job Costing and much more. This can be accomplished without prior programming knowledge.

Up to 1,000 records with a maximum of 20 headers (catagories) and 10 report formats (user defined) can be stored on a single diskette, Information can be sorted on any header, both ascending and decending in alpha/numeric field. Mathematical functions can be performed on any 2 fields to manipulate the information. Information can be searched on any header using >,<, = >, = <, =, and first letter. Mailing list format provided. Fast assembly language Sort, Search and Read routines. Many error protection devices provided. Put your application program together in minutes instead of hours.

Program Diskette and Instruction Manual........................ $100
Mailing List Program and Instruction Manual...................... $40

INVENTORY PROGRAM

2 disk drives, menu-driven program. Inventory categories include: Stock#, Description, Vendor ID, Class, Location, Reorder Pt., Reorder Qty., Qty. on Hand. All records can be entered, changed, updated, deleted, or viewed. Reports can be sorted in ascending/decending order by any category. There are 7 search reports (3 automatic). Calculates $ VALUE of Inventory and YTD, MTD, and period items sold, accumulates inventory over a 13-month period. Requires a 132-column, serial/parallel printer, total turnkey operation with bootstrap diskette.

Program Diskette and Instruction Manual........................ $140

PAYROLL PACKAGE*

2 disk drives, menu-driven program. Employee history include: Name, Address #, Address #2, City, State, Zip, Federal Exemption, State Exemption, Social Security #1, Date Employed, Dept. #, Code, Employee #, Status, Marital Status, Pay Rate, OT Rate, Vacation Rate, # Vacation Hours and Pension Plan. Program can generate weekly or biweekly payroll. Prints W-2, Qtr. Report, Pay Checks, Master and Current Files. Federal and State witholding taxes are built into program. Maintains a Cash Disbursement journal, accumulates payroll for a 53-week period. Generates numerous type of payroll reports. Allows data to be searched, sorted and edited. Prints Deduction Register and more. Maintain up to 125 Employees/Expenses for quick and easy Payroll. Numerous error protection devices provided.

Program Diskette and Instruction Manual........................ $240

*PLEASE SPECIFY STATE WHEN ORDERING

APARTMENT MANAGER

2 disk drives, menu-driven program written in assembly language and APPLESOFT II. All you will ever need to manage your apartment. Handles up to 6 Buildings with a maximum of 120 units each. Complete turnkey operation. Data categories include Apt. #, Type, Tenant Name, Pets, Children, Security Deposit, Pet Deposit, Pool Deposit, Misc. Deposit, Rent Allowances, Date Moved In, Vacancy Date, Referral, Condition of Apt., Damage Amt. and Comment Line. Search, sort, enter, edit and vacate tenants. Maintains MTD and YTD rent receipts as well as complete utility reports, rent lost by vacancies. Maintains Expenses, Vacated Tenants Report and much more.

Program Diskette and Instruction Manual........................ $325

PROFESSIONAL TIME AND BILLING

2 disk drive program written in assembly language and APPLESOFT II. Completely menu driven. Maintain all billing of clients and personnel. Generates and invoices. Numerous reports based on all types of criteria. Easy data entry for Rates, Clients, and Matters. Has Search, Sort, Change (on screen editing), View and Balance Forward. If you are a Job Contractor, Attorney, Accountant, General Consultant, or anyone that needs to charge for time, this program is a must. Complete turnkey operation. Many Reports are produced to aid in the Time Analysis Process.

Program Diskette and Instruction Manual........................ $325

ALL PROGRAMS REQUIRE 48K and APPLESOFT II ON ROM OR AND APPLE II PLUS. ALL SOFTWARE IS COMPATABLE WITH PASCAL SYSTEMS. PROGRAMS RUN FROM ANY PORT OF THE COMPUTER WITH SERIAL/PARALLEL PRINTERS. REQUIRES 1 DISK DRIVE UNLESS OTHERWISE NOTED.

SEND CHECK/MONEY ORDER or C.O.D. TO:

SOFTWARE TECHNOLOGY for COMPUTERS
P.O. BOX 428
BELMONT, MA 02178

(OR AVAILABLE FROM YOUR LOCAL DEALER)

Fun With OSI

A checker game is presented using Challenger C1P graphics and the polled keyboard techniques explained in William Taylor's series.

Les Cain
1319 North 16th Street
Grand Junction, Colorado 81501

I have been an avid fan of MICRO since August '79, and have enjoyed most of the articles. But everything is so serious, so come on let's have a few games. All work and no play makes Jack a dull boy.

What follows is a Checker game that uses Ohio Scientific graphics. As listed, the program is written to work on the 4P and 8P, but with slight modification it will also work on the 1P. The only differences are the video memory and the keyboard polling routines. The keyboard routines are explained adequately in the Graphics Manual and in other articles written on the Ohio Scientific system, so I won't go into these routines. The video memory on the 1P is 1K with decimal 32 between horizontal rows, while the 4P and 8P have 2K of memory with decimal 64 between horizontal rows.

I won't go into much detail as the listing explains the major program steps. Line 30 controls the video memory size and should be put in according to which computer is used. Line 310 reduces screen size to 32 × 32 on the 4P and 8P, and should be left out for the 1P. Lines 730 through 1220 are

```
10 FORI=1TO30:PRINT:NEXT
20 ST=53507:CC=128
30 REM
40 REM ST=53282 CC=64 FOR THE 1P
50 REM
60 SU=ST:CD=CC/2:B0=SU+12*CC+3
70 PRINTTAB(10);"## CHECKERS ##"
80 FORI=1TO10:PRINT:NEXT:FORT=1TO1000:NEXT
90 REM GET CHOICE OF MEN FOR PLAYER
100 REM Z1=PLAYER KING Z4=COMPUTER KING
110 REM
120 PRINT:PRINT:INPUT"DO YOU WANT RED OR BLACK";AN$
130 IF LEFT$(AN$,1)="R" THEN Z1=82:Z2=66:Z3=226:Z4=4:GOTO 160
140 IF LEFT$(AN$,1)<>"B" THEN 120
150 Z1=66:Z2=82:Z3=4:Z4=226:GOTO 160
160 PRINT:PRINT:PRINT"   INSTRUCTIONS:"
170 PRINT:PRINT:PRINT"MOVE FLASHING CHECKER BOARD TO MAN YOU"
180 PRINT"WANT TO MOVE BY U (UP), D (DOWN)"
190 PRINT"R (RIGHT), L (LEFT). PRESS CARRIAGE RETURN."
200 PRINT"THEN MOVE FLASING SQUARE TO THE POSITION"
210 PRINT"YOU WANT TO MOVE TO AND PRESS CARRIAGE"
220 PRINT"RETURN. IF YOU HAVE ANOTHER MOVE A FLASHING"
230 PRINT"CHECKER BOARD WILL INDICATE WHICH MAN TO"
240 PRINT"MOVE. IF NO MOVE CAN BE MADE PRESS SPACE"
250 PRINT"BAR. A LINE FEED ENDS GAME EARLY."
260 PRINT:PRINT:PRINT"     YOU ARE % ";AN$;" %"
270 PRINT:INPUT"TYPE C TO CONTINUE";AN$
280 FORI=1TO30:PRINT:NEXT
290 REM A&B STARTING PLACE FOR BOARD
300 A=SU:B=A+(3*CD)+3
310 POKE56900,0:REM TAKE OUT FOR 1P
320 DIM S(8,8),R1(4),R(4)
330 REM SETUP BOARD ARRAY
340 DATA 1,4,1,4,0,4,-1,4,4,1,4,0,4,-1,4,-1,15
350 FORI=0TO7:FORJ=0TO7:READX:IFX=15THEN370
360 S(I,J)=X:GOTO380
370 RESTORE:READS(I,J)
380 NEXTJ,I
390 B$="YOUR TURN":C$="TRY AGAIN"
400 REM
410 REM START BOARD DISPLAY
420 REM SQUARES FIRST
430 REM
440 FORI=1TO4:FORJ=1TO3
450 FORK=1TO4:FORL=1TO3
460 POKEA,161:POKEB,161
470 A=A+1:B=B+1:NEXTL
480 A=A+3:B=B+3:NEXTK
490 A=A+(CD-24):B=B+(CD-24):NEXTJ
500 A=A+(3*CD):B=B+(3*CD):NEXTI
510 REM BORDERS NEXT
520 A=SU+3:B=SU+(23*CD):C=B
530 F=SU+(3*CD):G=SU+23
540 FORI=1TO4:FORJ=1TO3
550 POKEA,135:POKEB,128
560 A=A+1:B=B+1:NEXTJ
570 A=A+3:B=B+3:NEXTI
580 FORI=1TO4:FORJ=1TO3
590 POKEF,136:POKEG,143
600 F=F+CD:G=G+CD:NEXTJ
610 F=F+(3*CD):G=G+(3*CD):NEXTI
620 REM CORNERS NEXT
630 POKESU+(23*CD),209:POKESU+23,207
640 G=-1:R(0)=-99
650 POKESU+(25*CD)-3,32:REM CLEAR CURSOR
660 REM POKE BOARD ARRAY
```

(continued)

(continued from page 6)

Dear Editor:

Here are BASIC pack program mods for KIM BASIC and Ohio Scientific BASIC-in-ROM.

The program by George Wells "SYM-1 BASIC Pack Program" [MICRO 25:19] was an exceptionally welcome one, since most of us find our desire for full documentation in conflict with the limited memory space in our computer. As with many programs which interact with Microsoft BASIC, it is readily adapted to other implementations on other 6502 machines.

Make the following hanges in the references to BASIC internal code:

Name	KIM address	Ohio Scientific Superboard
OUT.POINT	$7A	$7B
GET.RAM	$C0	$BC
IN.RAM.PNT	$256B	$A4A7
TEST.ALPHA	$2F33	$AD81

REM and DATA remain the same. The other three internal references occur sequentially in the program, beginning at $013B in the published program. These three, RST.BAS.PN, FIX.LIN.PN, and BASIC,WARM are replaced by one subroutine call. For KIM it is JSR $23EE and for Ohio Scientific Superboard it is JSR $A319. Due to the nature of the BASIC code at these points, this subroutine becomes an exit from the Pack program and you will be back in BASIC. The program's workings will be exactly as described in the article by Wells. To pack a program in BASIC workspace, exit to the monitor and GO at the start address of the program. Hit RETURN to fix the cursor and you can list your packed program, by typing LIST.

The KIM version tested was assembled at $020E and the Ohio Scientific version was assembled at $0222. The page one location should work in both cases, but I have not tried this.

Many thanks to George Wells for this practical program. Thanks to John Gibbins for the use of the Superboard to test the Ohio Scientific version.

Sean McKenna
64 Fairview Ave.
Piedmont, California 94610

```
670 GOSUB 1780
680 REM COUNT JUMPS AND SEE IF GAME OVER
690 IFC1=12THEN1670
700 IFP1=12THEN1720
710 FORI=1TO9:POKEB0+I,ASC(MID$(B$,I)):NEXT
720 REM POLLED KEYBOARD ROUTINE
730 Z=0
740 F1=1:F2=2
750 REM DISABLE CONTROL C POKE 530,1 FOR 1P
760 POKE2073,96:K=57088
770 LO=SU+(22*CD)+1:L1=0:U1=0
780 L2=L1-1:U2=U1-1
790 KI=187
800 REM THESE POKES AND PEEKS ARE DIFFERENT FOR 1P
810 REM GET MOVE
820 POKEK,32:F=PEEK(K)
830 PE=PEEK(LO)
840 POKELO,KI
850 FORT=1TO50:NEXTT
860 POKELO,32
870 FORT=1TO50:NEXTT
880 POKELO,PE
890 REM L FOR LEFT
900 IF F=64 THEN IF L1>0 THEN L1=L1-1:LO=LO-3
910 REM LINE FEED TO END GAME EARLY
920 IF F=16THEN POKE56900,1:END:REM NO POKE FOR 1P
930 REM CR TO INDICATE CHOICE MADE
940 IF F=8 THEN FORT=1TO100:NEXT:GOTO 1090
950 REM SPACE BAR TO INDICATE NO MOVE
960 POKEK,2:IF PEEK(K)=16 THEN 1270
970 POKEK,64:IF PEEK(K)=4ANDZ=0 THEN 740
980 POKEK,16:F=PEEK(K)
990 REM R FOR RIGHT
1000 IF F=32 THEN IF L1<7 THEN L1=L1+1:LO=LO+3
1010 REM U FOR UP
1020 IF F=4 THEN IF U1<7 THEN U1=U1+1:LO=LO-3*CD
1030 POKEK,2:IFPEEK(K)=16 THEN 1310
1040 POKEK,8:F=PEEK(K)
1050 REM D FOR DOWN
1060 IF F=64 THEN IF U1>0 THEN U1=U1-1:LO=LO+3*CD
1070 REM NO CHOICE GO AGAIN
1080 GOTO 820
1090 R1(F1)=L1:R1(F2)=U1:IF L2=L1 OR U2=U1 THEN 1230
1100 KI=161:REM SOLID SQUARE TO INDICATE CHOICE
1110 L2=L1:U2=U1
1120 IF F1=1 THEN F1=3:F2=4:GOTO 820
1130 E=R1(1):H=R1(2)
1140 A=R1(3):B=R1(4)
1150 IFS(E,H)=4ORS(A,B)<>0 THEN 1640
1160 S(A,B)=S(E,H):S(E,H)=0:IFABS(E-A)<>2THEN1260
1170 S((E+A)/2,(H+B)/2)=0:P1=P1+1:F2=4:F1=3
1180 Z=1
1190 REM CLEAR PROMPT AND UPDATE BOARD
1200 GOSUB 1870:GOSUB 1780
1210 REM CHANGE BACK TO CHECKERBOARD
1220 KI=187:GOTO 820
1230 A1=R1(F1):B1=R1(F2)
1240 IFS(A1,B1)<>0ORABS(A1-A)<>2ORABS(B1-B)<>2 THEN 1640
1250 E=A:H=B:A=A1:B=B1:I=I+15:GOTO1160
1260 IFB=7THENS(A,B)=2
1270 IFP=1THENGOSUB1870:GOTO650
1280 REM UPDATE BOARD
1290 GOSUB 1870:GOSUB 1780
1300 REM COMPUTER'S TURN
1310 FORX=0TO7:FORY=0TO7:IFS(X,Y)>-1THEN1340
1320 IFS(X,Y)=-1THENFORA=-1TO1STEP2:B=G:GOSUB1350:NEXTA
1330 IFS(X,Y)=-2THENFORA=-1TO1STEP2:FORB=-1TO1STEP2:GOSUB1350:NEXTB,A
1340 NEXTY,X:GOTO1510
1350 U=X+A:V=Y+B:IFU<0ORU>7ORV<0ORV>7THEN1400
1360 IFS(U,V)=0THENGOSUB1410:GOTO1400
1370 IFS(U,V)<0THEN1400
1380 U=U+A:V=V+B:IFU<0ORV<0ORU>7ORV>7THEN1400
1390 IFS(U,V)=0THENGOSUB1410
1400 RETURN
1410 IFV=0ANDS(X,Y)=-1THENQ=Q+2
1420 IFABS(Y-V)=2THENQ=Q+5
1430 IFY=7THENQ=Q-2
1440 IFY=0ORU=7THENQ=Q+1
1450 FORC=-1TO1STEP2:IFU+C<0ORU+C>7ORV+G<0THEN1490
1460 IFS(U+C,V+G)<0THENQ=Q+1:GOTO1490
1470 IFU-C<0ORU-C>7ORV-G>7THEN1490
1480 IFS(U+C,V+G)>0AND(S(U-C,V-G)=0OR(U-C=XANDV-G=Y))THENQ=Q-2
1490 NEXTC:IFQ>R(0)THENR(0)=Q:R(1)=X:R(2)=Y:R(3)=U:R(4)=V
1500 Q=0:RETURN
1510 IFR(0)=-99THEN1670
1520 R(0)=-99
1530 IFR(4)=0THENS(R(3),R(4))=-2:GOTO1550
1540 S(R(3),R(4))=S(R(1),R(2))
1550 S(R(1),R(2))=0:IFABS(R(1)-R(3))<>2THENGOSUB1780:GOTO690
1560 S((R(1)+R(3))/2,(R(2)+R(4))/2)=0:C1=C1+1
```

```
1570 X=R(3):Y=R(4):IFS(X,Y)=-1THENB=-2:FORA=-2TO2STEP4:GOSUB1610
1580 IFS(X,Y)=-2THENFORA=-2TO2STEP4:FORB=-2TO2STEP4:GOSUB1610:NEXTB
1590 NEXTA:IFR(0)<>-99 THENR(0)=-99:GOTO 1530
1600 GOTO 670
1610 U=X+A:V=Y+B:IFU<0ORU>7ORV<0ORV>7THEN1630
1620 IFS(U,V)=0ANDS(X+A/2,Y+B/2)>0THENGOSUB1410
1630 RETURN
1640 GOSUB1870:FORI=1TO9:POKEB0+I,ASC(MID$(C$,I)):NEXT
1650 E1=E1-1:FORT=1TO500:NEXTT:GOSUB1870:IFP<>1THEN740
1660 GOTO670
1670 PRINT"CONGRATULATIONS YOU WON"
1690 REM RESTORE SCREEN SIZE AND CONTROL C
1710 POKE56900,1:POKE2073,173:END
1720 PRINT"I WON TOUGH LUCK"
1730 REM
1740 POKE56900,1:POKE2073,173:END
1750 REM
1760 REM POKE BOARD ARRAY
1770 REM
1780 D1=SU+(CD+1):FORJ=7TO0STEP-1:FORI=0TO7
1790 IFS(I,J)=4THENPOKED1,161:GOTO1850
1800 IFS(I,J)=0THENPOKED1,32:GOTO1850
1810 IFS(I,J)=1THENPOKED1,Z1:GOTO1850
1820 IFS(I,J)=-1THENPOKED1,Z2:GOTO1850
1830 IFS(I,J)=2THENPOKED1,Z3:GOTO1850
1840 IFS(I,J)=-2THENPOKED1,Z4
1850 D1=D1+3:NEXT:D1=D1+168:NEXT:RETURN
1860 REM CLEAR PROMPT
1870 FORI=1TO40:POKEB0+I,32:NEXT:RETURN
```

the polled keyboard routines for the 4P and 8P. There are sufficient REMARKS to change the PEEKs and POKEs, so that they match the 1P's keyboard routine. Line 1710 and 1740 should be POKE 530,0 for the 1P. The routines from line 1780 to 1850 POKE to the screen the present location of the player's and the computer's checkers.

One final note: if you get tired of playing the game you can use the display to adjust your TV or monitor.

Les Cain is employed by Bureau of Land Management as a Civil Engineering Technician where he received training in time-sharing microcomputers.

MICRO™

WANTED!
Good Articles and Good Photos
MICRO Pays Very Well!

MICRO is committed to covering all of the 6502 systems. To do this well, we need a variety of articles on each system. We can always use more high-quality articles relating to AIM, SYM, KIM, Apple, Atari, PET/CBM, and Ohio Scientific systems. We are especially interested in good articles which apply to 6502 systems in general.

Because we plan to use more illustrations than formerly, we encourage authors to "think pictorially" and to send us good line drawings and black and white photos.

We are also looking for black and white photos which might stand alone, with a brief caption. Photos of 6502 systems in unusual business or professional environments would be especially welcome. Photos used independently of articles will be paid for separately.

For details write or telephone:

Editorial Department
MICRO
P.O. Box 6502
Chelmsford, MA 01824
617/256-5515

Semi-Annual Index

June – November 1980* (Issues 25 – 30)

Articles

Issue/Page

AIM

Share Your AIM Programs — 25:23
Jody Nelis

AIM-65 File Operations — 26:61
Christopher J. Flynn

Satellite Tracking with the AIM-65 — 27:13
C.R. MacCluer

Loading KIM-1 Tapes to AIM — 28:19
Larry P. Gonzalez

Compact — 28:25
Steve Bresson

Tiny PILOT for the AIM — 28:59
Larry Kollar and Carl Gutekunst

An Improved Morse Code Receive Routine and Interface — 29:23
Marvin L. DeJong

Biorhythm: An AIM BASIC Programming Exercise — 29:51
P.E. Burcher

AIM 65 File Operations: Writing Text Files with BASIC — 30:65
Christopher J. Flynn

APPLE

A Little Plus For Your Apple II — 25:7
Craig Peterson

APPLE II Integer BASIC Program List by Page — 25:37
Dave Partyka

BASIC and Machine Language Transfers with Micromodem II — 25:47
George Dombrowski

TRACER: A Debugging Tool for the APPLE II — 25:59
R. Kovacs

Zoom and Squeeze — 26:37
Gary B. Little

Data Statements Revisited — 27:7
Virginia Lee Brady

Better Utilization of Apple Computer Renumber and Merge Program — 27:17
Frank D. Chipchase

Solar System Simulation with or without an Apple II — 27:33
David A. Partyka

Applesoft Floating Point Routines — 27:53
R.M. Mottola

Business Dollars and Sense in Applesoft — 27:65
Barton M. Bauers, Jr.

Creating Shape Tables, Improved! — 28:7
Peter A. Cook

A Versatile Hi-Res Function Plotter for the Apple II — 28:49
David P. Allen

Mean 14: A Pseudo-Machine Floating Point Processor for the Apple II — 28:67
R.M. Mottola

PRINT USING for Applesoft — 29:14
Gary A. Morris

Paged Printer Output for the APPLE — 29:47
Gary Little

Cassette Label Program — 29:65
Dawn E. Ellis

Step and Trace for the APPLE II Plus — 30:61
Craig Peterson

ATARI

Introducing the Atari 800 — 25:35
William L. Colsher

Atari Notes — 27:57
William L. Colsher

A Versatile Hi-Res Function Plotter for the Atari 400 & 800 — 30:47
David P. Allen

KIM

VISA—KIM — 26:47
Joel Swank

A "Stop-on-Address" Routine for KIM — 29:30
R. MacDonald

OHIO SCIENTIFIC

Put Your Hooks Into OSI BASIC — 25:15
Edward H. Carlson

Hypocycloids on the 540 — 25:57
E.D. Morris

Challenger II Communications — 26:53
Peter Koski

Interface of OSI C1P With Heath Printer — 27:47
William L. Taylor

A C1P and H14 System, Part 2 — 28:30
William L. Taylor

An OSI Cheep Print — 29:7
Thomas Berger

An Ultra-Fast Tape Storage System — 30:11
John E. Hart

Ohio Scientific Users: Stop those S'ERRORS — 30:37
E.D. Morris, Jr. and Tim Finkbeiner

PET

Lower Case Lister — 25:11
Rev. James Strasma

PET-16 — 25:49
Rev. James Strasma

'Stop That PET' - Update — 25:64
George R. Gaukel

Hello, World — 26:31
John Sherburne

Son of Screen Print — 27:61
Kenneth Finn

Auto-Run-Save, Y-t Plotter, Canary for the PET — 28:14
Werner Kolbe

Define Your Own Function Key on PET — 29:19
Werner Kolbe

For Multiple File Tape Backups — 29:36
G.R. Boynton

Self-modifying PET Programs — 30:29
P. Kenneth Morse

SYM

SYM-1 BASIC Pack Program — 25:19
George H. Wells, Jr.

Slide Show for the SYM — 25:53
David P. Kemp

SYM-1 Memory Search and Display — 26:7
Nicholas Vrtis

SYM-Bell — 30:17
Randy Sebra

(continued)

*MICRO's volume year runs from June through May. Issue numbers span volumes consecutively, from MICRO's first bimonthly issue (Oct./Nov. 1977) to the current monthly issue (No. 32).

GENERAL

6502 Resource Update 25:65
Dr. William R. Dial

Sorting Revealed 26:13
Richard C. Vile, Jr.

Variable Lister 27:19
Ray Cadmus

Additions to Tiny Pilot 27:21
Bob Applegate

Nth Precision Add & Subtract With Adjusted Processor Status 27:27
Lawrence R. Golla

BCD Input to a 6502 Microprocessor 27:68
Richard Saltero

XREFER 28:34
Joel Swank

Undedicating a dedicated Microcomputer 29:27
David N. Borton

Tiny Pilot Complemental (Co-Pilot) 29:32
Robert Schultz

Hexadecimal Printer 29:57
LeRoy Moyer

Programming with Pascal 29:59
John P. Mulligan

How to Use the Hooks 30:7
Richard Williams

John Conway's Game of Life Using Display Devices With Automatic Scrolling 30:53
Theodore E. Bridge

Departments

	Issue/Page
The MICRO Software Catalog *Mike Rowe**	25:71 / 26:71 / 27:71 28:73 / 29:73 / 30:72
6502 Bibliography *Dr. William R. Dial*	25:75 / 26:75 / 27:75 28:76 / 29:76 / 30:76
MICRO Club Circuit *Mike Rowe**	25:68 / 26:68 / 27:25 28:47 / 29:34 / 30:72
MICROScope	27:31 / 28:57 / 29:49 / 30:33
Microprocessors in Medicine: The 6502 *Jerry W. Froelich, M.D.*	29:56 / 30:36
Up From the Basements *Jeff Beamsley*	27:59 / 29:72 / 30:51
PET Vet *Loren Wright*	28:48 / 29:39 / 30:27
Letterbox	26:60 / 27:56 / 29:6
Editorial *Robert M. Tripp*	25:5 / 26:5 / 27:5 / 28:5 / 29:5 / 30:5

*Mike Rowe is a pseudonym for material prepared by MICRO's staff.

This January issue of the Ohio Scientific Small Systems Journal contains two articles, both based on machine language (assembly) programs.

The first article is an implementation of two system memory tests. Both of the tests will run on all Ohio Scientific computers, video or serial based.

The second feature this month is a compact implementation of the game of *Life* for Ohio Scientific 4PMF computers. The same program may be used on 8PDF systems by changing the origin in line 170 to * = $317E. *Life* was developed by Professor John Conway at the University of Cambridge and was first described in the "Mathematical Games" column of the October, 1970 *Scientific American.*

Happy New Year!

Small Systems Journal
1333 S. Chillicothe Road
Aurora, Ohio 44202

Memory Tests

In this month's Small Systems Journal we are presenting two memory tests. The first test is a bit rotation test, while the second test uses a pseudo-random byte generator. The algorithms for each test will be explained in another part of the article. Either test will operate on all Ohio Scientific computers.

The most appropriate time to use a memory test is when new memory is installed in the computer. For example, expanding a Superboard II with 4K RAM to a system with 8K RAM.

On the other hand, there is no reason not to run an occasional memory test on an "up and running" computer. It gives you great peace of mind to have memory testing perfectly before you start any new major project.

```
                 ;
                 ;BIT ROTATION MEMORY TEST
                 ;
                 ;
                 ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
                 ;                                                           ;
                 ; TO RUN TEST, PRESET                                       ;
                 ;    'MSTART' = FIRST ADDRESS TO BE TESTED (0,1) - LO,HI    ;
                 ;    'MEND'   = LAST ADDRESS TO BE TESTED +1 (2,3) - LO,HI  ;
                 ;    START EXECUTION AT 'ENTRY'                             ;
                 ;                                                           ;
                 ; IF MEMORY TESTS OK, SERIES OF 'X' WILL BE PRINTED         ;
                 ;                                                           ;
                 ; IF ERROR, 'E' WILL BE PRINTED AND PROGRAM WILL STOP       ;
                 ; RESET COMPUTER AND EXAMINE LOCATIONS:                     ;
                 ;    'MPOINT' = MEMORY ADDRESS OF ERROR (4,5) - LO,HI       ;
                 ;    'DATA'   = CORRECT TEST DATA (6)                       ;
                 ;    'ERROR'  = ERROR DATA (7)                              ;
                 ;                                                           ;
                 ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
                 ;
                 ;
                 ;PAGE ZERO LOCATIONS
0000=                   MSTART=0          FIXED START ADDRESS
0002=                   MEND  =2          FIXED ENDING ADDRESS
                 ;
0004=                   MPOINT=4          TEST POINTER
0006=                   DATA  =6          CORRECT DATA
0007=                   ERROR =7          ERROR DATA
                 ;
                 ;
0240              ENTRY *=$240            TEST ENTRY ADDRESS
                 ;
0240 A2FF               LDX #$FF
0242 9A                 TXS               SET STACK POINTER
0243 E8                 INX               INDEX FOR OSI VIDEO
0244 D8                 CLD
0245 208B02       MTEST JSR SETPNT        SET 'MPOINT'
                 ;
0248 A000               LDY #0            CLEAR INDEX
024A 98                 TYA
024B 9104         MCLR  STA (MPOINT),Y    CLR TEST BLOCK
024D 209402             JSR ADJPNT        ADJ 'MPOINT'
0250 90F9               BCC MCLR          CONTINUE BLOCK
0252 208B02             JSR SETPNT        DONE, RESET 'MPOINT'
                 ;
0255 A900         RTST1 LDA #0
0257 8506               STA DATA
0259 B104               LDA (MPOINT),Y    STILL 0 ?
025B C506               CMP DATA
025D D022               BNE MERR          NO, ERROR
                 ;
025F 38                 SEC
0260 2A           RTST2 ROL A             ROTATE BIT THRU (A)
0261 9011               BCC RTST3         ALL BITS NOT DONE
                 ;
0263 A9FF               LDA #$FF          ALL BITS DONE
0265 9104               STA (MPOINT),Y    SET HIGH
0267 209402             JSR ADJPNT        ADJ 'MPOINT'
026A 90E9               BCC RTST1         NOT DONE W/BLOCK
                 ;
026C A958               LDA #'X           BLOCK DONE, MARK OK
026E 20A502             JSR OUT
0271 4C4502             JMP MTEST         REDO TEST
                 ;
0274 8506         RTST3 STA DATA          SAVE (A)
0276 9104               STA (MPOINT),Y    WRITE BIT PATTERN
0278 B104               LDA (MPOINT),Y    GET DATA
027A C506               CMP DATA          SAME?
027C D003               BNE MERR          NO, ERROR
027E 18                 CLC               OK, CLR FOR NEXT 'ROL'
027F 90DF               BCC RTST2 (BRA)
                 ;
0281 8507         MERR  STA ERROR         ERROR DATA
0283 A945               LDA #'E           MARK ERROR
0285 20A502             JSR OUT
0288 4C8802             JMP *             STOP EXECUTION
                 ;
                 ;
028B A500         SETPNT LDA MSTART       SETUP 'MPOINT'
028D 8504               STA MPOINT
028F A501               LDA MSTART+1
0291 8505               STA MPOINT+1
0293 60                 RTS
                 ;
0294 48           ADJPNT PHA              SAVE (A)
0295 E604               INC MPOINT        ADJ 'MPOINT'
0297 D002               BNE *+4           NO PAGE CROSS
0299 E605               INC MPOINT+1      ADJ PAGE
                 ;
029B A504               LDA MPOINT        TEST IF BLOCK DONE
029D C502               CMP MEND
029F A505               LDA MPOINT+1
02A1 E503               SBC MEND+1
02A3 68                 PLA               RESTORE (A)
02A4 60                 RTS               C=SET=BLOCK DONE
                 ;
```

```
02A5 9DCED0  OUT    STA $D0CE,X       FOR OSI VIDEO
02A8 E8             INX
02A9 8D01FC         STA $FC01         FOR OSI SERIAL
02AC 60             RTS

                     END
```

```
                    ;
                    ;PSEUDO-RANDOM BYTE MEMORY TEST
                    ;
                    ;
                    ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
                    ;                                                                 ;
                    ;  TO RUN TEST, PRESET                                            ;
                    ;      'MSTART' = FIRST ADDRESS TO BE TESTED (0,1) - LO,HI        ;
                    ;      'MEND'   = LAST ADDRESS TO BE TESTED +1 (2,3) - LO,HI      ;
                    ;      START EXECUTION AT 'ENTRY'                                 ;
                    ;                                                                 ;
                    ;  IF MEMORY TESTS OK, SERIES OF 'X' WILL BE PRINTED              ;
                    ;                                                                 ;
                    ;  IF ERROR, 'E' WILL BE PRINTED AND PROGRAM WILL STOP            ;
                    ;  RESET COMPUTER AND EXAMINE LOCATIONS:                          ;
                    ;      'MPOINT' = MEMORY ADDRESS OF ERROR (4,5) - LO,HI           ;
                    ;      'DATA'   = CORRECT TEST DATA (6)                           ;
                    ;      'ERROR'  = ERROR DATA (7)                                  ;
                    ;                                                                 ;
                    ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
                    ;
                    ;
                    ;PAGE ZERO LOCATIONS
0000=                      MSTART=0          FIXED START ADDRESS
0002=                      MEND  =2          FIXED ENDING ADDRESS
                    .
0004=                      MPOINT=4          TEST POINTER
0006=                      DATA  =6          CORRECT DATA
0007=                      ERROR =7          ERROR DATA
                    ;
0008=                      RANDOM=8          PSEUDO-RANDOM DATA WORD
                    .
                    ;
0240                       *=$240            TEST ENTRY ADDRESS
                    ;
0240 A2FF           RNDTST LDX #$FF
0242 9A                    TXS               SET STACK POINTER
0243 E8                    INX               INDEX FOR OSI VIDEO
0244 D8                    CLD
                    ;
0245 209F02         RBTEST JSR SETPNT        SET 'MPOINT'
0248 A509                  LDA RANDOM+1      SAVE RANDOM
024A 48                    PHA
024B A508                  LDA RANDOM
024D 48                    PHA
                    ;
024E 208602         RT.1   JSR RNBYTE        GET RANDOM DATA
0251 9104                  STA (MPOINT),Y    WRITE TO BLOCK
0253 20A802                JSR ADJPNT        NEXT LOCATION
0256 90F6                  BCC RT.1          CONTINUE
                    ;
0258 68                    PLA
0259 8508                  STA RANDOM        RESTORE RANDOM
025B 68                    PLA
025C 8509                  STA RANDOM+1
025E 209F02                JSR SETPNT        SET 'MPOINT'
0261 4C6902                JMP RT.3
                    ;
0264 20A802         RT.2   JSR ADJPNT        NEXT LOCATION
0267 B015                  BCS RT.4          DONE
0269 208602         RT.3   JSR RNBYTE        GET RANDOM DATA
026C 8506                  STA DATA
026E B104                  LDA (MPOINT),Y    GET MEM DATA
0270 C506                  CMP DATA          SAME?
0272 F0F0                  BEQ RT.2          YES, CONT
                    ;
0274 8507                  STA ERROR         NO, ERROR DATA
0276 A945                  LDA #'E
0278 20B902                JSR OUT
027B 4C7B02                JMP *             STOP EXECUTION
                    ;
027E A958           RT.4   LDA #'X           BLOCK DONE, MARK OK
0280 20B902                JSR OUT
0283 4C4502                JMP RBTEST        REDO TEST
                    ;
                    ;
0286 A409           RNBYTE LDY RANDOM+1      RANDOM=(RANDOM*5)+1 W/OVRFLO
0288 A508                  LDA RANDOM
028A 0608                  ASL RANDOM
028C 2609                  ROL RANDOM+1      RANDOM*2
028E 2608                  ROL RANDOM        WRAP OVERFLOW
0290 2609                  ROL RANDOM+1      (RANDOM*2)*2
                    ;
0292 38                    SEC               SET FOR +1
0293 6508                  ADC RANDOM
0295 8508                  STA RANDOM
0297 98                    TYA
0298 6509                  ADC RANDOM+1
029A 8509                  STA RANDOM+1      RANDOM=((RANDOM*2)*2)+RANDOM+1
029C A000                  LDY #0            CLEAR (Y)
029E 60                    RTS
                    ;
                    ;
029F A500           SETPNT LDA MSTART        SETUP 'MPOINT'
02A1 8504                  STA MPOINT
02A3 A501                  LDA MSTART+1
02A5 8505                  STA MPOINT+1
02A7 60                    RTS
                    ;
02A8 48             ADJPNT PHA               SAVE (A)
02A9 E604                  INC MPOINT        ADJ 'MPOINT'
02AB D002                  BNE *+4           NO PAGE CROSS
02AD E605                  INC MPOINT+1      ADJ PAGE
                    ;
02AF A504                  LDA MPOINT        TEST IF BLOCK DONE
02B1 C502                  CMP MEND
02B3 A505                  LDA MPOINT+1
02B5 E503                  SBC MEND+1
02B7 68                    PLA               RESTORE (A)
02B8 60                    RTS               C=SET=BLOCK DONE
                    ;
02B9 9DCED0         OUT    STA $D0CE,X       FOR OSI VIDEO
02BC E8                    INX
02BD 8D01FC                STA $FC01         FOR OSI SERIAL
02C0 60                    RTS
                    ;
                           .END
```

Bit Rotation Test

The bit rotation test uses the following five step algorithm:

1) Initially, set all memory under test equal to zero.
2) If location under test is not zero, error.
3) Rotate single bit through byte location (1, 2, 4, 8, etc.) and test if OK; if not OK, error.
4) Set location equal to FF_{16} and advance to next location.
5) Continue test pass at Step 2.

This test is very useful for locating address line short circuits. This is because every byte to be tested should be zero prior to the rotation test. If there is an address line short, setting the current byte under test (in Step 4) to hex FF will cause another byte higher in memory to also become hex FF. This test is also handy for easy identification of a totally defective memory chip since only one bit is examined at a time.

Pseudo-Random Test

The pseudo-random data test uses the following test algorithm:

1) Write pseudo-random byte to location under test.
2) Advance to next location and continue with Step 1 until all memory under test has been written with the pseudo-random sequence.
3) Restart at initial test location.
4) Using same pseudo-random sequence, verify each location has not changed; if changed, error.
5) After all memory under test has been verified, restart at Step 1 with new pseudo-random sequence.

The pseudo-random test has the advantage, with respect to the bit rotation test, that it is very sensitive to data "pattern sensitivity". This means that "typical" RAM data is usefully approximated. A disadvantage of this type of test is that many complete passes of the test must be completed to assure good memory.

Running the Tests

Each test is a very basic "bare bones" program. The starting and ending addresses of the memory to be tested must be preset before running the test. Upon completion of every test pass, an "X" will be displayed at your terminal device. If an error should occur, an "E" will be displayed and the test will stop.

To determine the cause of the error, the computer must be reset and the contents of memory at locations 4, 5, 6, and 7 examined. This will contain the address of the error (4 and 5), the expected data (6) and the data actually found in memory (7).

A couple of examples should clarify the test procedure. If memory between hex 0300 and hex 07FF inclusive is to be tested on a C4P, C1P, etc., the procedure would be as follows: (note: <CR> means RETURN)

1) enter the desired memory test into memory
2) Reset your computer and type M
3) Type — .0000/00<CR>03 <CR>00<CR>08
4) Type — .0240 G

The test should begin execution and after several seconds, if all is well, begin to display a series of "X"s.

To demonstrate an error condition, use the following procedure:

1) enter the bit rotation test
2) reset your computer and type M
3) type — .0000/00<CR>FE <CR>00<CR>FF
4) type — .0240 G

Using this method (testing ROM) an error ("E") should occur immediately. Reset your computer and examine memory using this sequence:

type — .0004/ display — 00
<CR>display — FE
<CR>display — 00
<CR>display — A2

```
10                 ;        LIFE PROGRAM FOR C4P MF
20                 ;
30  DE00=                   CTLREG=$DE00
40  DF00=                   KEY   =$DF00
50                 ;
60  0001=                   ROW   =$01
70  0005=                   START =$05
80  0040=                   COLUMN=$40
90                 ;
100 00FF=                   HOLD  =$FF
110 00FE=                   COUNT =$FE
120 00FC=                   TEMP  =$FC
130 00FA=                   VIDSCR=$FA
140 00F8=                   VIDBUF=$F8
150 00F0=                   SEED  =$F0
160                ;
170 327E                    *=$327E
180 327E A905               LDA #START ;   INITIALIZE RANDOM NUMBERS
190 3280 85F0               STA SEED
200 3282 85F4               STA SEED+4
210                ;
220 3284 8D00DE             STA CTLREG ;   TURN ON COLOR, WIDE SCREEN
230                ;
240 3287 204233 NEW         JSR INIT   ;   RESET POINTERS TO BEGIN
250                                    ;   BLOCK MOVE
260                ;
270 328A B1F8   COPY        LDA (VIDBUF),Y ;   COPY BUFFER CONTENTS
280 328C 91FA               STA (VIDSCR),Y ;   TO COLOR SCREEN
290 328E 205533             JSR MOVEUP
300 3291 D0F7               BNE COPY
310                ;
320 3293 204233             JSR INIT   ;   RESET POINTERS TO BEGIN
321                                    ;   NEW GENERATION
330                ;
340 3296 A901               LDA #ROW   ;   CHECK FOR EXIT KEY
350 3298 8D00DF             STA KEY    ;   BEFORE WE GET TOO
360 329B AD00DF             LDA KEY    ;   FAR ALONG
370 329E 2940               AND #COLUMN
380 32A0 F001               BEQ SAME
390 32A2 60                 RTS
400                ;
410 32A3 A5FA   SAME        LDA VIDSCR ;   SETUP TEMPORARY POINTER
420 32A5 85FC               STA TEMP   ;   FOR CURRENT CELL
430 32A7 A5FB               LDA VIDSCR+1
440 32A9 85FD               STA TEMP+1
450                ;
460 32AB A900               LDA #0     ;   CLEAR NEIGHBOR COUNT
470 32AD 85FE               STA COUNT  ;   FOR THIS CELL
480                ;
490 32AF 203733             JSR LOOKUP ;   READ CURRENT CELL
500                ;
510 32B2 A5FE               LDA COUNT  ;   REMEMBER IF IT STARTED
520 32B4 48                 PHA        ;   OUT DEAD OR ALIVE
530                ;
540 32B5 A901               LDA #1     ;   LOOK IN ALL EIGHT
550 32B7 201E33             JSR AHEAD  ;   DIRECTIONS AND
560 32BA A940               LDA #64    ;   COUNT THE NEIGHBORS
570 32BC 201E33             JSR AHEAD  ;   AS WE GO
580 32BF A901               LDA #1
590 32C1 202A33             JSR BEHIND
600 32C4 A901               LDA #1
610 32C6 202A33             JSR BEHIND
620 32C9 A940               LDA #64
630 32CB 202A33             JSR BEHIND
640 32CE A940               LDA #64
650 32D0 202A33             JSR BEHIND
660 32D3 A901               LDA #1
670 32D5 201E33             JSR AHEAD
680 32D8 A901               LDA #1
690 32DA 201E33             JSR AHEAD
700                ;
710 32DD 68                 PLA        ;   RESTORE CURRENT CELL
720 32DE 08                 PHP        ;   STATUS--ASSUME ITS DEAD
730 32DF A90E               LDA #14
740 32E1 28                 PLP
750 32E2 48                 PHA
760 32E3 D024               BNE ALIVE
```

```
                         ;
32E5 A5FE          LDA COUNT  ;   COMES TO LIFE IF EXACTLY
32E7 C903          CMP #3     ;   THREE NEIGHBORS
32E9 D028          BNE UPDATE
                   ;
32EB 68    RANDOM  PLA        ;    IT'S NOT DEAD (YET!)
32EC 38            SEC
32ED A5F1          LDA SEED+1 ;    GET A RANDOM COLOR
32EF 65F4          ADC SEED+4 ;    IN ITS HONOR
32F1 65F5          ADC SEED+5
32F3 85F0          STA SEED
32F5 8A            TXA
32F6 48            PHA
32F7 A204          LDX #4
32F9 B5F0  MOVE    LDA SEED,X
32FB 95F1          STA SEED+1,X
32FD CA            DEX
32FE 10F9          BPL MOVE
3300 68            PLA
3301 AA            TAX
3302 A5F0          LDA SEED
3304 0901          ORA #1
3306 48            PHA
3307 D00A          BNE UPDATE ;   PUT IT AWAY
                   ;
3309 A5FE  ALIVE   LDA COUNT  ;   IF ALREADY ALIVE, IT
330B C903          CMP #3     ;   STAYS THAT WAY IF
330D 3004          BMI UPDATE ;   2, 3, OR 4 NEIGHBORS
330F C906          CMP #6
3311 30D8          BMI RANDOM
3313 68    UPDATE  PLA        ;   UPDATE CURRENT CELL
3314 91F8          STA (VIDBUF),Y
                   ;
3316 205533        JSR MOVEUP ;   MOVE TO NEXT CELL
                   ;
3319 D088          BNE SAME   ;   KEEP GOING
331B 4C8732        JMP NEW
                   ;
331E 18    AHEAD   CLC        ;   MOVE AHEAD IN TEMPORARY
331F 65FC          ADC TEMP   ;   POINTER
3321 85FC          STA TEMP
3323 9002          BCC *+4
3325 E6FD          INC TEMP+1
3327 4C3733        JMP LOOKUP
                   ;
332A 38    BEHIND  SEC        ;   MOVE BACK IN TEMPORARY
332B 85FF          STA HOLD   ;   POINTER
332D A5FC          LDA TEMP
332F E5FF          SBC HOLD
3331 85FC          STA TEMP
3333 B002          BCS *+4
3335 C6FD          DEC TEMP+1
                   ;
3337 B1FC  LOOKUP  LDA (TEMP),Y
3339 290F          AND #15    ;   SEE IF TEMPORARY CELL
333B C90E          CMP #14    ;   IS ALIVE AND COUNT IT
333D F002          BEQ *+4
333F E6FE          INC COUNT
3341 60            RTS
                   ;
3342 A933  INIT    LDA #$33   ;    INITIALIZE POINTERS
3344 85F9          STA VIDBUF+1
3346 A97E          LDA #$7E   ;    BUFFER @$337E
3348 85F8          STA VIDBUF
334A A9E0          LDA #$E0
334C 85FB          STA VIDSCR+1
334E A000          LDY #$00   ;    SCREEN @$E000
3350 84FA          STY VIDSCR
3352 A208          LDX #8     ;    COUNT 8 PAGES
3354 60            RTS
                   ;
3355 C8    MOVEUP  INY        ;    MOVE POINTERS FORWARD
3356 D005          BNE OK
3358 E6F9          INC VIDBUF+1
335A E6FB          INC VIDSCR+1
335C CA            DEX
335D 60    OK      RTS
```

This tells you that at location $FE00_{16}$ a zero was expected to be in memory, but $A2_{16}$ was found to be in that location instead.

A similar error could be generated using the pseudo-random test, but the test data at location 6 would vary.

A complete assembly listing of both the bit-rotation test and the pseudo-random data test is given. Please note that these tests are both assembled at hex 0240. If you wish to have both tests in memory at the same time, one or the other must be re-assembled.

The Game of Life

Life is perhaps the simplest simulation program around today. *Life*, as the name implies, deals with a life and death generation process based on a primitve cell which, in this implementation, translates to a location of video memory. If a cell starts out alive, that is, in color, then it will remain so if it is adjacent to 3, 4, or 5 other live cells. If a cell starts out dead, that is, black and white, then it will remain so unless adjacent to exactly three live cells. Thus, while the program is running, the screen shows a color pattern of the living and dead cells as each generation passes. As the program is set up, approximately 1.5 generations will be generated per second. However, if the random number generation is replaced by a single default color, execution is considerably quicker. (In fact, the random number generator was added to slow the program enough to observe the pattern changes.) Before running, the initial pattern may be entered into locations \$337E-\$3B7D which correspond directly to the color video at \$E000-\$E7FF.

To run the program, enter it as listed in the assembler, assemble it using "A3", and execute by typing "!GO 327E". The display will then show a pattern of live and dead cells changing more than once a second. To stop the program at any point, press the CONTROL key. You may then make any observations you wish on your "culture". To re-enter your program, again type "!GO 327E" and the program will continue with the next generation.

MICRO

Software Catalog: XXVIII

Name: **Disk Copy/Disk Space in ROM**
System: Basic Apple II or Apple II Plus
Memory: 32K or 48K
Language: Assembly (ROM Chip)
Hardware: Mountain Hardware's ROMPLUS Board Disk II with 3.2 or 3.3 DOS
Description: Plug this ROM into your ROMPLUS board and this utility will be a keyboard command away from your immediate use. Two disk utility programs on one ROM chip. DISK COPY will duplicate a disk with one disk drive, two disk drives with one controller, or two drives and two controllers. Initialization and volume number change are selectable. When DISK SPACE is activated it will read the track bit map on a disk and display the number of free sectors and bytes still available for use. The DISK SPACE routine will also allow you to read or write to any sector, on any track, from assembly language. Will operate with: DOS 3.2 or 3.3, 32K or 48K, Int, FP, LS, II or II Plus. Applesoft Renumber/Merge in ROM also available.
Copies: Many
Price: $49.95
Includes: User Documentation and ROM Chip
Author: **Frank D. Chipchase**
Available: **Soft CTRL Systems**
P.O. Box 599
West Milford, NJ 07480

Name: **Single Disk Copy Utility**
System: OSI C1, C4
Hardware: Disk with 24K - 32K
Description: A copier for most users. Copies OS65D disks with or without assembler and monitor, OS-MDMS system and OS-MDMS data disks, based on the directory. Copies only the first sector on a track except for the multi-sectored tracks in the operating systems (6 and 12). Simple to use with prompts. 32K system copies 8 tracks at a time. Makes multiple copies. Does not copy track 0, something we should all understand how to do.
Price: $9.95
Author: **Kenneth Madell**
Available: **Earthship**
P.O. Box 489
Sussex, New Jersey 07461

Name: **32K Apple Pi 'Life'**
System: Apple II
Memory: 32K RAM Minimum
Language: Integer BASIC, Machine Language
Hardware: Cassette Drive or Disk II
Description: Hi-Res grid size up to 64 x 64. Speed from 10 to 60 generations/minute at 64 x 64 grid size. Over 100 pre-defined objects as given in various computer magazines. Options and sub-options are chosen by capital letters given below. The Modify option allows Build constellation, Create object, Delete constellation, Erase object, and Get object from grid. The Grid storage option allows Get grid from available storage (30 grids for 48K & DOS) or Put grid in storage. An Inspection option allows Objects, Constellations, or Grids to be automatically displayed for inspection or deletion. A Plot option allows Objects, Constellations, Hollow and Solid blocks, Lines, and Random points to be plotted on the grid. The Transfer option allows tables to be Loaded or Saved and grid storage to be Read or Written, using either Cassette or Diskette. Finally, there are Zero grid and Continue 'Life' generation options.
Price: $10.00 for cassette or $12.50 for diskette.
Author: **Harry L. Pruetz**
Available: **Microspan Software**
709 Caldwell St.
Yoakum, TX 77995

Name: **STOCKPLOT; COMPLOT**
System: PET/CBM
Memory: 8K to 32K
Language: BASIC
Hardware: PET/CBM, disk, 2022 printer.
Description: Plots stock and commodity price histories. Allows data base update by user to keep plot current. One year's price history included on disk. Data smoothing and digital filtering provide trend reversal indication.
Price: $50 for Dow Jones Industrials. Other stock prices on request.
Author: **Samuel P. Cook**
Available: **Cook Compusystems**
309 Lincolnshire
Irving, Texas 75061

Name: **Disk Utilities 2.1**
System: Apple II
Memory: 48K and ROM Applesoft
Language: Applesoft and Machine Language
Hardware: Disk II
Description: A menu driven collection of five utility programs. All programs work with DOS 3.2 and DOS 3.2.1. Copy—creates backup copies of your important program and data disks. Compare—verifies that a diskette is a copy of another. Certify—verifies that all used sectors on a diskette can be read. Statistics—reports the amount of used and unused space on a diskette both in number of sectors and as a percentage of the total diskette. Patch—a program for the advanced user who wants to access any byte on a diskette. With this program you can 'undelete' a deleted file, protect a bad sector from access, remove or add control characters in file names and much more.
Price: $19.95 on diskette, with manual.
Author: **Hal Clark**
Available: **On-Going Ideas**
RD #1 Box 810
Starksboro, VT 05487

Name: **Hex-ASCII Memory Dump**
System: Apple II or Apple II Plus
Memory: Any
Language: 6502 Machine Code
Hardware: Apple II or Apple II Plus
Description: A utility program designed for use in the software development environment. The ASCII conversion output makes locating textual data in memory extremely easy. Runs stand alone or interfaces to either BASIC through POKEs and CALLs. Output to video or printer. In interactive mode it supports paging, scrolling or stop-start. Four data entry formats and five control functions. Small, occupies only 512 bytes. Completely relocatable with monitor move command or BLOAD to address. An invaluable aid for the serious programmer.
Copies: Just Released
Price: $15.95 on disk includes disk and documentation
Author: **Richard E. Rettke**
Available: **RER Software**
1757 Acorn Ct.
Menasha, WI 54952

Name: **Modem Magic**
System: Apple II/Apple II Plus
Memory: 48K (ROM Applesoft)
Language: Applesoft/Machine Language
Hardware: D.C. Hayes Micromodem, Disk II
Description: Five programs to automatically download and upload data and programs to remote computers (such as a CBBS or another Apple). Special routines to strip out unnecessary 'formatting' blanks from BASIC programs (to minimize transmission time) and to properly handshake with slow receiving computers.
Copies 25
Price: $40.00
Author: **Gary Little**
Available: **Gary Little**
#101-2044 West Third Avenue
Vancouver, B.C., Canada
V6J 1L5

Name: **Reading Comprehension**
System: Apple II Plus
Memory: 16K
Language: BASIC
Hardware: Applesoft in ROM
Description: Has five short stories or poems that you must read. Speed of reading is set by your age. Then you answer 10 questions about the story or poem. Can change data on tape—on tape.
Copies: As Needed
Price: $10.00
Author: **Jim and Lois Willis**
Available: **Jim Willis**
1300 Hinton
West Monroe, LA 71291

Name: **Financial Management System**
System: Apple II or Apple II Plus
Memory: 48K with ROM Applesoft
Language: Applesoft
Hardware: Disk II, Printer Optional
Description: A disk-based home accounting system with unlimited files. User defined macros allow 1-3 keystroke entrys and automatic tax coding. Audit all files by tax code with monthly and year-to-date totals. Printer listings of disk files, balance, reconcile, search, audit, and macro lists.
Price: $29.95 on disk includes diskette and complete documentation.
Author: **D.R. Jarvis**
Available: **D.R. Jarvis Computing**
1039 Cadiz Dr.
Simi, CA 93065

Name: **The Environment Life Dynamic**
System: Apple II
Memory: 48K
Language: Applesoft
Hardware: Apple II, Disk II
Description: This disk gives you a long, intense look at environment, its effects, aids to improving it, and ways of transcending or transforming it. The disk deals with differentiating between a good and bad environment, identifying positive environmental characteristics, oppressd aspects of yourself, and finding rare individuals who would be perfect for being part of a truly beautiful lifestyle/space. *Lightning Bugs, Turn 'Em Loose!*, and the incomparably unique *Jungle Safari* are 3 of the included Hi-Res (shape table) games. You'll love them *and* learn from them.
Copies Many
Price: $15.95 includes disk, game card.
Author: **Avant-Garde Creations**
Available: **Avant-Garde Creations**
P.O. Box 30161 MCC
Eugene, OR 97403

Name: **2516/2716 EPROM Programmer**
System: SYM-1
Memory: 8K
Language: BASIC
Hardware: 3rd 6522 VIA
Description: With just $5 in spare parts, this powerful programmer package will 1) program EPROMS, 2) verify EPROM is cleared, 3) copy EPROM to memory, 4) display EPROM data, 5) display SYM memory, 6) enter new data into SYM memory, and more. Any number of bytes can be programmed (read) from (into) any area of memory. Instructions included.
Copies: New Release
Price: $15
Author: **Jack Gieryic**
Available: **Jack Gieryic**
2041 138th Ave. N.W.
Andover, MN 55303

These listings are free. In order to obtain a free listing, however, software producers or distributors must follow the format used here. The description part of the listing may NOT exceed 12 typeset lines.

Listings are published in the order in which they are received. However, only one entry per company is accepted for publication in any single month. When more than one entry is submitted, the company submitting the entry must establish the relative priorities. If the software product or its price change after an entry is submitted, it is the responsibility of the company to notify MICRO prior to publication.

Name: **Editing Terminal ROM**
System: OSI C1P, Superboard
Memory: No RAM Used
Language: 6502 Machine Code
Hardware: None Special
Description: This ROM Monitor provides a smart terminal program, BASIC program line recall/edit, ASCII file transfer between terminal/ROM BASIC/disk BASIC, program uploading/down loading, easy transfer of programs between tape and disk, serial output driver, keyboard corrected to typing format, screen clear. All features available at power on.
Copies Just Released
Price: $59.95 for ROM and 11 page manual.
Author: **Leo Weeks**
Available: **Micro Interface**
3111 So. Valley View Blvd. Suite I-101
Las Vegas, NV 89102

Name: **Video Games 2**
System: OSI C1, C2, C4, C8 BASIC in ROM
Memory: 8K
Language: BASIC
Hardware: None Special
Description: Video Games 2 consists of three games: Gremlin Hunt, Gunfight and Indy 5000. Gremlin Hunt is an arcade-style game for one to three players. Players try to run over 'gremlins' with their tanks. Gunfight is a duel of mobile artillery for one or two. Indy 5000 is a race game for one or two. Color and sound for machines so equipped.
Copies Just Released
Price: $15 on cassette tape, ppd.
Author: **Mike Bassman**
Available: **Orion Software Associates**
147 Main Street
Ossining, NY 10562

Name: **FIFO Morse Typewriter**
System: AIM 65, 1K or 4K
Memory: Needs 500 Bytes
Language: Assembly
Hardware: Simple keying circuit for transmitter included.
Description: For ham radio buffs. FIFO operation, keyboard always active. Display automatically switches from keyboard to output buffer so that operator doesn't lose track. Delete, speed adjust features included.
Copies: Just Released
Price: $9.95. Object file cassette, source available.
Author: **Alan M. Davis**
Available: **A. M. Davis**
RFD #2130, Route 106
Syosset, New York 11791

"COMPUTERS ' R' US"

A CONSUMER COMPUTERS SUBSIDIARY

UNBEATABLE MAIL ORDER DISCOUNTS

apple computer
Authorized Dealer

NEW! CALL FOR AVAILABILITY AND PRICES.

$925 FOR 16K

48K FOR ONLY $1049

apple ///

APPLE II OR APPLE II PLUS

APPLE COMPUTER PERIPHERALS

DISK II DRIVE & CONTROLLER CARD With DOS 3.3, List $645	529
DISK II DRIVE & CONTROLLER card	485
DISK II DRIVE ONLY	425
GRAPHICS TABLET	655
SILENTYPE PRINTER w/int. card	515
SSM AIO SERIAL/PARALLEL kit	155
SSM AIO assembled & tested	190
SYMTEC LIGHT PEN SYSTEM	215
SYMTEC SUPER SOUND GENERATOR	225
SVA 8 INCH DISK CONTROLLER CARD	335
VERSA WRITER DIGITIZER SYSTEM	215
VIDEX VIDEOTERM 80 COLUMN CARD	315
VIDEX VIDEOTERM w/graphics ROM	335
LOBO DISK DRIVE ONLY	385
LOBO DRIVE w/controller card	485
HAYES MICROMODEM II	319
DAN PAYMAR lower case kit	55

APPLE COMPUTER INTERFACE CARDS

PARALLEL PRINTER int. card	145
COMMUNICATION CARD w/conn. cable	185
HI-SPEED SERIAL int. card	145
LANGUAGE SYSTEM with PASCAL	425
CENTRONICS PRINTER int. card	185
APPLESOFT II FIRMWARE card	149
INTEGER BASIC FIRMWARE card	149

MOUNTAIN HARDWARE ACCESSORIES
A Division Of
Mountain Computer

APPLE CLOCK/CALENDAR card	225
SUPERTALKER SD200 SPEECH SYNTHISIZER SYSTEM	245
ROMPLUS w/keyboard filter	185
INTROL/X-10 BSR REMOTE CONTROL SYSTEM	245
INTROL/X-10 controller card only	165
ROMWRITER SYSTEM	155
MUSIC SYSTEM(16 voices/stereo)	465
A/D-D/A 16 CHANNELS	319
EXPANSION CHASSIS (8 slots)	555

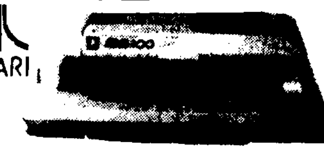
APPLE ADD-ONS

CORVUS 10 MEGABYTE HARD DISK DRIVE SYSTEMw/pwr supply	4395
CORVUS CONSTELLATION	595
16K MEMORY UPGRADE KIT (TRS-80, APPLE II, SORCERER)	60
ABT NUMERIC INPUT KEYPAD (specify old or new kybrd)	115
ALF MUSIC SYNTHISIZER	235
BRIGHTPEN LIGHTPEN	32
GPIB IEEE-488 (1978) Int.	259
ARITHMETIC PROCESSOR card	335
SPEECHLINK 2000 (64 Word Vocab.)	215
M&R SUP-R-MOD TV MODULATOR	30
MICROSOFT Z-80 SOFTCARD SYSTEM w/CP/M & MICROSOFT BASIC	299
MICROWORKS DS-65 DIGISECTOR	339
LAZER lower case adapter	50
M&R SUPER TERMINAL 80 column card	335

APPLE II or APPLE II PLUS SOFTWARE

PASCAL with LANGUAGE SYSTEM	425
FORTRAN for use with LANGUAGE SYSTEM	165
CP/M for use with MICROSOFT Z-80 SOFTCARD (incl.)	299
DOS 3.3	49
THE CONTROLLER General Business System	519
THE CASHIER Retail Management & Inventory System	199
APPLEWRITER Word Processor	65
APPLEPOST MAILING list system	45
APPLEPLOT Graph & Plot Sytem	60
DOW JONES PORTFOLIO EVALUATOR	45
APPLE CONTRIBUTED VOLUMES 1 thru 5 w/manuals	30
VISI-CALC by PERSONAL SOFTWARE	120
DESKTOP/PLAN by DESKTOP COMPUTERS	85
CCA DATA MANAGEMENT SYSTEM By PERSONAL SOFTWARE	85
APPLEBUG ASSEMBLER/DISASSEMBLER	75
APPLE DOS TOOL KIT	65

ATARI



16K FOR $799

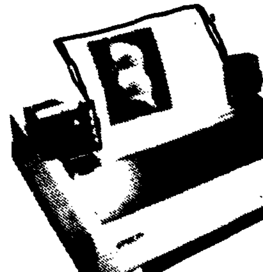
ATARI 800
PERSONAL COMPUTER SYSTEM

ATARI ACCESSORIES

400 COMPUTER	479
820 PRINTER (40 col)	459
810 DISK DRIVE	559
410 Program Recorder	59
815 DUAL DISK DRIVE	1199
822 THERMAL PRINTER (40 col)	369
825 PRINTER (80 col. imp.)	795
850 INTERFACE MODULE	175
ATARI 16K RAM MODULE	155
LIGHT PEN	65
ACOUSTIC MODEM (CAT)	169
COMPUTER CHESS	35
SPACE INVADERS	19
STAR RAIDERS	49
SUPER BREAKOUT	35
3-D TIC-TAC-TOE	35
VIDEO EASEL	35
MUSIC COMPOSER	49

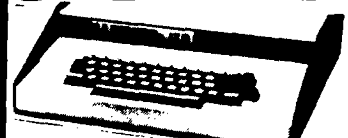
PRINTERS

ANADEX DP-8000	775
ANADEX DP-9500	1350
BASE 2	599
CENTRONICS 737	825
MPI 88-T	699
PAPER TIGER IDS-440 w/graphics	895
NEC SPINWRITER	2550
TRENDCOM 200	519
SILENTYPE w/int	515
EPSON TX-80 w/graphics	729
EPSON MX-80 132 col	620



EPSON TX-80

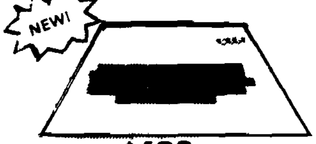
OHIO SCIENTIFIC



$699

C4P
8K ROM BASIC
8K RAM EXPANDABLE TO 96K
32x64 UPPER & LOWER CASE
256x512 GRAPHICS POINTS
PROGRAMMABLE TONES
ANALOG INPUTS
C4PMF (1 DISK DRIVE) 1599

NEW!



$429

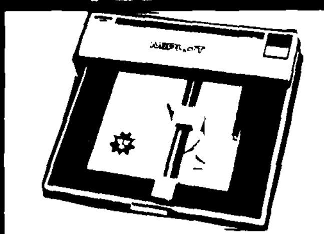
CIP MOD II
8K ROM BASIC
8K RAM EXPANDABLE TO 32K
COLOR EXPANSION
48 LINE DISPLAY EXPANSION

SOFTWARE

	Cassette	Disk
SPACE INVADERS	19	29
SARGON II	30	35
FORTH	N/A	69
OS 65-D V3.3	N/A	79
MDMS PLANNER	N/A	100
GRAPHICS I	N/A	35
DAC I	N/A	45
ASSEMBLER/EDITOR	40	N/A
EXTENDED MONITOR	20	N/A
PASCAL & FORTRAN (4P & 8P only)		450

When ordering please specify system.

PLOTTERS



only $1095

WATANABE MIPLOT

for more info please call or write

VIDEO MONITORS

LEEDEX VIDEO 100	129
SANYO 9" B&W	165
SANYO 15" B&W	245
PANACOLOR 10" COLOR	329
NEC 12" HI-RES COLOR	875
NEC 12" LO-RES COLOR	399
NEC 12" GREEN PHOSPHER(P31)	239

$129

LEEDEX VIDEO 100

- FAST DELIVERY
- LOW PRICES
- COURTEOUS SERVICE
- KNOWLEDGEABLE STAFF
- LARGE VARIETY

IN CALIFORNIA, OR FOR BACKORDER OR TECHNICAL INFO CALL: **(714) 698-8088**

TOLL FREE ORDER LINE: 1-800-854-6654

CREDIT CARD USERS PLEASE READ TERMS OF SALE IN ORDERING INFORMATION

ORDERING INFORMATION: Phone Orders invited using VISA, MASTERCARD, AMERICAN EXPRESS, or bank wire transfers. VISA & MC credit card service charge of 2%. AE credit card service charge of 5%. Mail orders may send charge card number (include expiration date), cashier's check, money order or personal check (allow 10 business days to clear.) Please include a telephone number with all orders. Foreign orders (excluding Military PO's) add 10% for shipping and all funds must be in US dollars. Shipping, handling and insurance in U.S. add 3%. California residents add 6% sales tax. Our low margins prohibit us to send COD or on account. All equipment subject to price change and availability. Equipment is new and complete with manufacturer warranty. We ship most orders within 2 days. Order desk hours are Monday thru Saturday 9-5 PST. Send for FREE 1981 Catalog.

WE ARE A MEMBER OF THE BETTER BUSINESS BUREAU AND THE CHAMBER OF COMMERCE

RETAIL STORE PRICES MAY DIFFER FROM MAIL ORDER PRICES.

PLEASE SEND ORDERS TO: CONSUMER COMPUTERS MAIL ORDER CRU Division 8314 PARKWAY DRIVE, GROSSMONT SHOPPING CENTER NORTH, LA MESA, CALIFORNIA, 92041

MICRO

6502 Bibliography: Part XXVIII

Dr. William R. Dial
438 Roslyn Avenue
Akron, Ohio 44320

797. Call - Apple 3, No. 5 (June, 1980)

Wagner, Roger, "Text Output on the Apple II," pg. 7-17.
A major article on how text is output to the Apple screen including an example machine language program which will scroll either page 1 or page 2 of the text both up and down.

Reynolds, William, III, "Applesoft Program Splitter," pg. 19-22.
Moves program lines inside a program in order to protect Apple HGR page(s) or for insertion of OP codes inside a program.

Golding, Val, "Into Integer," pg. 31-32.
A discussion of the merits of Integer Basic for the Apple together with a program for Integer Basic String Array.

Curtiss, Dick, "CP/M for Apple II," pg. 33.
How to install the Control Program/Monitor (CP/M) on an Apple. Requires the new Z80 plug-in card. This is said to open the door for a whole new library of sophisticated programs available to the CP/M users.

Huelsdonk, Bob, "Making Basic Behave: Part III," pg. 35-37.
A continuation of the development of an Apple utility: this installment continues with the base program, adding input, edit, and disk file routines.

Reynolds, Lee, "EXEC Files on the Apple II,", pg. 39-40.
Take advantage of the very useful EXEC function on the Apple.

Hertzfeld, Andy, "Init and Switch," pg. 43.
A Hex dump routine in machine language for the Apple II.

Golding, Val, "Write Apple," pg. 44.
A neat trick to save space by BSAVEing Applesoft integers or reals to disk. Also a routine for disabling the stop list at full page for the Apple disk Catalog command.

Hewitt, Jay, "Disk I/O in Pascal," pg. 48.
Two programs, one in Basic and one in Pascal, to show how this routine translates from one system to the other.

Golding, Val J., "Applesoft Text Screen Dump," pg. 51.
A short routine for the Apple.

798. Interface Age 5, No. 6 (June, 1980)

Leary, Richard A., "Mixed Interfaces,", pg. 108-112.
Interfacing the 6502 to a Z-80.

Wetzel, Ken, "A Break Service Routine for a KIM-1 with a Teletype," pg. 132-135.
Routine allows listing of all the internal registers of the 6502 CPU whenever it encounters a breakpoint.

799. Dr. Dobb's Journal 5, Issue 6, No. 46 (June/July, 1980)

Bach, Stephen E., "ROM Byte-Finder for the Apple II," pg. 26.
Routine prints all 256 values and addresses in the ROM in the Apple.

Bach, Stephen E., "Improvements to Apple User Interface," pg. 56.
Improvements for the Renumbering program in DDJ 42.

800. Byte 5, No. 6 (June, 1980)

Renbarger, John, "A Telephone-Dialing Microcomputer," pg. 140-170.
A simple hardware addition and software for the KIM-1 to provide telephone dialing on Touch-Tone systems.

801. Kilobaud Microcomputing, No. 42 (June, 1980)

Hadeishi, Mits, "Additional Indexing Features," pg. 20.
Routine for the PET.

Prentice, Al, "A File Sorting Program and its Diary," pg. 34-40.
Describes the development of a File-Sorting program for the Apple.

Monsour, Fred, "Cook's Memory Test for the 6502," pg. 178-179.
This program tests 8k of RAM in less than 3 seconds.

802. Peek(65) 1, No. 6 (June, 1980)

Carroll, Michael B., "Auto-Loading Machine Language Casssette Tapes," pg. 7-8.
Here are two methods for OSI systems.

Morton, Ian A., "Large RAM and Long Programs," pg. 14.
Some hints concerning inputting from cassette and keyboard on OSI micros.

803. Softside: Apple Edition (June, 1980)

Truckenbrod, Joan, "Apple Graphics," pg. 12-14.
A tutorial on Hi-Res graphics explaining how figures are moved in various directions.

Pelczarski, Mark, "Program for Your Disk System," pg. 33-34.
A special "hello" program for the Apple disk.

Pelczarski, Mark, "High Resolution Drawings from The Magic Paint Brush," pg. 44-48.
Program is an adaptation of an earlier program to operate in high resolution graphics on the Apple.

Blackwood, Brian and George, "Intimate Instructions in Integer BASIC," pg. 49-54.
Lessons III, IV and V of the continuing tutorial for the Apple.

804. Creative Computing 6, No. 6 (June, 1980)

Ault, Robin, "More Error Trapping Input," pg. 10.
Discussion pertinent to Apple and Applesoft programs.

Tubb, Philip, "Apple Music Synthesizer," pg. 74-83.
With a computer controlled synthesizer it is claimed that almost any instrument or sound can be duplicated.

Anon., "Sound Apple Hint," pg. 89.
How to hear your Apple a little more clearly by adding an outboard speaker.

Heuer, Randy, "Apple Hi-Res Graphics Made Easy with the VersaWriter," pg. 92-93.
A review of a drawing board for the Apple.

Rogowski, Steve, "Graphics Goodies: The Case for Polar," pg. 98-103.
Graphics tutorial including a listing in Pascal for the Apple Turtlegraphics.

Simoni, Richard T., Jr., "The Intricate Graphs of the Polar Functions," pg. 104-107.
A computer program for the Apple to graph polar functions.

Vile, Richard C., Jr., "Apple Kaleidoscopes," pg. 110-113.
Methods of developing kaleidoscope routines.

805. OSI Users Independent Newsletter, No. 4 (June, 1980)

Curley, Charles, "Notes of the OS 65U," pg. 1-2.
POKEs of use to OS 65U video system users, other addresses of importance, etc.

Hooper, Phil, "String Finder Routine with Wild Card," pg. 4-5.
A string finder extension to the extended OSI monitor which allows for don't care bytes.

Curley, Charles, "BASIC," pg. 5.
Several "little known" facts about 65U BASIC.

Curley, Charles, "Disassembly," pg. 5-6.
A quick and dirty mini assembler for the OSI system.

806. Stems from Apple 3, No. 6 (June, 1980)

Stein, Dick, "Integer Renumber," pg. 4.
To renumber Integer programs change them to Applesoft, renumber with existing routines and then change back to Integer.

Robin, Neil A., "ROM/PROM Testing Program," pg. 9.
A nearly foolproof method for calculating ROM/PROM checksums.

Hoggatt, Ken, "Random Number Analysis," pg. 10.
A routine for the Apple to test your random number generator at several numbers and compare results.

Erickson, Bret and Hoggatt, Ken, "Pascal Math Program," pg. 11-13.
A listing in Pascal for the Apple is given.

Stein, Dick, "File Cabinet Changes," pg. 13.
More changes for this oft-upgraded program.

Good, Bob and Reed, Ron, "Recdemo," pg. 14-15.
Takes data Input from keyboard, writes 3 records to disk, reads 3 records from disk and displays data; in Apple Pascal.

807. G.R.A.P.E. (July, 1980)

Trusty, Doug and Lawson, Steve, "GRAPE Hi-Res Writing," pg. 4.
Several routines for the Apple including Trusty Scroller, Hi-Res Writer, and English, Greek, Hebrew Type Fonts.

808. Rubber Apple Users Newsletter 3, No. 6 (July, 1980)

Anon., "Taking the GRRR Out of Graphics," pg. 3.
A tutorial on Hi-Res graphics, creating pictures, saving them and recovering pictures.

Anon., "Tabbing with Apple Peripherals," pg. 5.
With printer driver routines for parallel, serial and other printer systems.

809. MICRO No. 25 (June, 1980)

Peterson, Craig, "A Little Plus for Your Apple II," pg. 7-9.
A routine for the Apple II to provide the features of the Apple II Plus.

Carlson, Edward H., "Put Your Hooks Into OSI BASIC," pg. 15-17.
Extend your OSI BASIC in ROM? Very easy. This article tells how.

Nelis, Jody, "Share Your AIM Programs," pg. 23-33.
A routine for combining the AIM disassembler output with comments.

Partyka, Dave, "Apple II Integer BASIC Program List by Page," pg. 37-39.
This program simplifies the viewing of an Apple listing by printing it page by page.

Dombrowski, George J., Jr., "BASIC and Machine Language with the Micromodem II," pg. 47-48.
A program to send programs over phone wires using the Apple II and the D.C. Hayes Micromodem II.

Strasma, Rev. James, "PET-16," pg. 49-51.
For those PET owners who have envied the Sweet-16 software of the Apple, here is PET-16.

Kemp, David P., "Slide Show for the SYM," pg. 53-56.
An Apple to SYM Picture Translator. Permits a SYM with visible memory to use the Apple cassette tapes to put on a slide show.

Kovacs, R., "TRACER: A Debugging Tool for the Apple II," pg. 59-61.
Use this program to make your Apple STEP/TRACE routines more useful.

Dial, Dr. Wm. R., "6502 Resource Update," pg. 65-66.
A list of magazines which contain information about the 6502 microprocessor field on a reasonably regular basis. Includes addresses, subscription fees, etc.

Staff, "MICRO Club Circuit," pg. 68-69.
Another installment of 6502-related clubs.

Rowe, Mike (MICRO Staff), "The MICRO Software Catalog: XXI," pg. 71-73.
Fourteen new programs for 6502 systems are reviewed.

Dial, Dr. Wm. R., "6502 Bibliography: Part XXI," pg. 75-77.
One hundred new references to 6502-related articles.

810 The Apple Barrel 3, No. 5 (June/July, 1980)

Kramer, Mike, "File Cabinet Stuffer," pg. 7-9.
A program for the Apple to allow examination, entering or changing values in the File Cabinet program.

Meador, Lee, "Disk Operating System—Part 3, DOS 3.2 Disassembly," pg. 10-14.
The third installment of a comprehensive series on the Apple DOS.

811. FortWorth Area Apple User Group Newsletter 1, No. 9 (July, 1980)

Stringham, David, "File Cabinet: Sorting Two Files," pg. 1-4.

Running File Cabinet on two Apple Disk Drives and using two files.

Anon., "Macro POP," pg. 7-8.
This assembly language program pops a 16-bit address for Apple.

Meador, Lee, "Disassembly of DOS 3.2: Part VI," pg. 10-14.
The sixth installment of this major series on Apple DOS.

812. The Seed 2, No. 7 (July, 1980)

Kelley, Jim, "Quick Printer II Routine,", pg. 6.
A short Machine Language routine for the Apple.

Webber, Stan, "Getting Fancy with Formats," pg. 11-12.
A number of different types of formatting and examples of their implementation.

Crossman, Craig, "Apple Tricks," pg. 15.
How to fix an Apple program that will not list and self-destructs when it has finished running.

Suitor, Dick, "The False Read of the 6502," pg. 17-18.
A discussion of timing signals and reading data.

Steinmetz, Lori, "DOS Patch to Double the Speed of Most Disk Operations," pg. 18.
Short machine language patch to increase disk operation speed and other notes on disk operation.

Anon., "Apple Hi-Res Routines," pg. 19.
A list of entry points to program machine level Hi-Res graphics.

813. Compute, Issue 4 (May/June, 1980)

Lock, Robert and DeJong, Marvin L., "Keeping Up the Payments," pg. 19..
Three Loan programs that should run on the AIM, Apple, Atari or PET.

Kushnier, Ron, "Computers—Boring, Boring, Boring," pg. 27.
Use of the PET to print out temperatures using an analog/digital converter and a temperature sensor.

Isaacs, Larry, "Inside Atari BASIC," pg. 31-34.
Information on how Atari BASIC stores programs in memory.

Deal, Elizabeth, "Big Files on a Small Computer," pg. 42-46.
Program demonstrating a way of reducing storage requirements by a factor of eight—for the PET.

Davis, Harvey, "Algebraic Input for the PET," pg. 58.
Description and listing of a useful routine for the PET.

Straley, Ron, "PET Data Copier," pg. 59.
Here is a routine that will copy any PET data file or data tape so you will have a backup data copy.

Butterfield, Jim, "Cross-Reference for the PET," pg. 63.
Notes on the 2040 disk system, reading a BASIC Program as a file, and detailed syntax analysis, etc.

Greenberg, Gary, "PET GET with Flashing Cursor," pg. 77-78.
Program listing which permits the user to use the GET statement instead of the INPUT statement on the PET.

Herman, Harvey B., "PETting with a Joystick," pg. 89-90.
Install a joystick on your PET to save the keyboard from wear in games, etc.

Johnson, Chuck, "PET and the Dual Joysticks," pg. 90.
How to install two joysticks on your PET.

Thornburg, David D., "Made in the Shade: An Introduction to 'Three Dimensional' Graphics on the Atari Computers," pg. 97-98.
A tutorial on Atari Graphics. Setting and adjusting shades of color.

Lindsay, Len, " 'Enter' with Atari," pg. 103.
How to insert or append program segments on the Atari Computer.

Conrad, Tom, "Block Access Method Map for a Commodore 2040 Disk Drive," pg. 104-106.
This map program will allow you to see where your files are allocated. Save and Delete files and observe the allocation technique.

McCarthy, C.A., "CHEEP PRINT: Hard Copy for Soft Cost," pg. 111-114.
CHEEP PRINT is a PET program, mostly in BASIC, and serves both to list the host program while it is under development, and as data output software after the program has been completed.

814. Southeastern Software Issue No. 19 (July, 1980)

McClelland, George, "Binary Search of Names File," pg. 5-9.
Here is a program to do a binary search of the NAMES FILE program given in previous SES Newsletters.

815. L.A.U.G.H.S. 2, No. 2 (July, 1980)

Connelly, Pat, "Animation," pg. 3-4.
General tutorial discussing Apple Hi-Res graphics.

816. Compute II, Issue 2 (June/July, 1980)

DeJong, Marvin L., "Some A/D and D/A Conversion Techniques," pg. 5-12.
6502 techniques applicable to AIM, KIM, SYM systems. Suggested hardware, A/D, D/A driver program for 6522-based timing and driver program for KIM-1 interface.

Butterfield, Jim, "Some Routines from Microsoft BASIC," pg. 13.
Addresses identifying the start of the area in which a large number of routines on the KIM, SYM, AIM and OSI Micros.

Zumchak, Gene, "Nuts and Volts," pg. 17-19.
A good discussion of 6502 read/write timing. Interfacing to an existing system or a do-it-yourself prototype is not difficult as long as you understand and consider timing.

Beach, Edward B., "Fast Random Numbers for the 6502," pg. 24.
A short (two instructions) -Fast (ten microseconds) routine which produces an 8-bit random number (not pseudorandom).

Day, Michael E., "Part 2: RS232 Communications," pg. 27-29.
A list of the definitions of the RS232-C signals in the order of pin numbers. With detailed descriptions.

Butterfield, Jim, "BASIC Memory Map (Page 0)", pg. 33.
List showing assignments on zero page for KIM, SYM, AIM and OSI C2-4P.

Olsen, Rodger, "Creating Data Files on Tape with OSI Computers," pg. 41-45.
A 'how to' article on files on tape. With listings in BASIC for several routines realated to the subject.

Zumchak, Gene, "SYM High Speed Tape," pg. 55-56.
How to improve the SYM tape read.

Nazarian, Bruce, "KIM Rapid Memory Load/Dump Routine," pg. 57-60.
A routine which works well for mass entering of long programs from a hex dump, where you can tell at a glance where any errors in your program are.

Herman, Harvey B., "KIM-1 Tidbits," pg. 60.
Several programs to facilitate the use of the KIM.

817. The Abacus II, 2, Issue 7 (July, 1980)

Anon., "Color Generation in Hi-Res: Part 2," pg. 3-5.
Part two on Apple Hi-Res graphics.

Robbins, Greg, "Extended Menu," pg. 5-7.
This menu program for the Apple Disk scrolls commands across the bottom of the screen and provides copy service as well as the full range of disk commands.

Anon, "IAC APNOTE—Out of Memory Errors," pg. 9.
Notes on the Applesoft error message.

Davis, Jim and Freeman, Larry, "Relocatable Control Character Detector," pg. 14-15.
Program to detect control characters on the Apple except for CTRL-H, CTRL-M, CTRL-U or ESCape characters.

Anon., "IAC APNOTE: Applesoft Array Eraser," pg. 18.
A short program to demonstrate how to erase Appleso Arrays.

818. Kilobaud Microcomputing, No. 43 (July, 1980)

Baker, Robert W. "PET Pourri," pg. 7-9.
Describes several new products for the PET; Programming Ideas and Tips; and discussion of programming style.

Prentice, Al, "A File Sorting Program and Its Diary," pg. 44-52.
Second and concluding part completes the sorting routine and further explores the Apple DOS.

Derfler, Frank J., "Dial-Up Directory," pg. 68-70.
Software programs for Modem-type communications.

Pytlik, William F., "PET Pen," pg. 84-86.
Part 2 of 3 installments gives listing and construction details for a light pen for the PET.

Mazur, Jeffrey G. "Add Handshaking to Apple's High Speed Interface," pg. 136-138.
This modification allows the High Speed Interface board to handshake properly with printers. Does not require a software patch.

819. Nibble No. 4 (July, 1980)

Harvey, Mike, "Forecasting and Planning with FAST," pg. 7.
Listing and documentation for FAST, a simulation and modeling tool for testing different strategies in the direct sales of multiple products and services, for the Apple.

Laird, Alexander, "Super Weaver!", pg. 63.
Super Weaver is a simulated 8-harness loom in Apple Hi-Res Graphics.

Staff, "Table Printing Made Simple," pg. 19.
A formatting routine for the Apple.

Staff, "Dynamic Array Dimensioning," pg. 19-20.
Automatically adjust program memory allocation to the specific mix of table specifications.

Connolly, Rick, "Designing a Simple 'Pointer' Subroutine," pg. 31.
A tutorial covering the development of a pointer routine which will return a key number to the main program of the Apple.

Harvey, Mike, "Build the Two-Tape Control Unit," pg. 35-37.
Hardware and Software article to increase flexibility of your tape storage.

Mottola, R.M., "Hi-Res Packing for the Apple II,' pg. 41.
How to keep your Applesoft program out of the way of the Hi-Res graphics screen on the Apple.

Harvey, Mike, "TRAC Screen Only Update," pg. 53.
Modification of the TRAC printer versions to run with Screen output only.

820. Personal Computing 4, No. 7 (July, 1980)

Walker, Alan, "Cassette Tape Labels," pg. 33.
A program for the Commodore PET to print labels for cassette tapes.

Brown, DeWitt, "Amortization Tables," pg. 73-75.
Program to print customized amortization tables on the Apple.

821. Appleseed Newsletter 2, No. 2 (July, 1980)

Hyde, Bill, "Subroutine to Print the Lo-Res Screen," pg. 2.
Listing to print out Lo-Res apple graphics screens on a character printer using a different character for each color.

Wright, Don, "Applesoft Program List Formatter," pg. 5.
A new and useful formatter program.

822. MICRO No. 26 (July, 1080)

Vrtis, Nicholas, "SYM-1 Memory Search and Display," pg. 7-11.
Add two new commands to your SYM monitor to locate any string in memory and provide a means to display data as ASCII when desired.

Sherburne, John, "Hellow, World," pg. 31-35.
An analog interface for the PET.

Little, Gary B., "Zoom and Squeeze," pg. 37-38.
A short program for the Apple II which makes it easier to edit BASIC programs.

Swank, Joel, "Viza-KIM," pg. 47-50.
A KIM Monitor program to display system parameters at each step. Uses the 6502 interrupt handling routine.

Koski, Peter, "Challenger II Communications," pg. 53-58.
Turn your OSI with a 502 CPU board into a 'standard' communications terminal.

Flynn, Christopher J., "AIM 65 File Operations," pg. 61-66.
Programs to solve the problem of missing file access statements in AIM BASIC.

Rowe, Mike (Staff), "The MICRO Software Catalog: XXII," pg. 71-73.
Thirteen new programs are reviewed.

Dial, Wm. R., "6502 Bibliography: Part XXII," pg. 75-77.
About 95 new references on the 6502.

823. The River City Apple Corps Newsletter (July, 1980)

Sethre, Tom, "Roots," pg. 5-6.
Looping in Apple machine language focussing on the specific task of duplicating the BASIC function of PRINTing a string.

Huffman, David G., "Shape Definition Conversion Table," pg. 8-9.
A Hi-Res tutorial for Apple graphics.

824. Creative Computing 6, No. 7 (July, 1980)

Carpenter, Chuck, "Apple-Cart," pg. 150-152.
How to get better sound out of your Apple, notes on booting disks, software over the phone, new assemblers and simulators.

Blank, George, "Outpost: Atari," pg. 154-155.
Notes on information resources for the Atari microcomputer; refers to Compute, Softside: Atari, and a new magazine, IRIDIS.

825. SoftSide: Apple Edition (July, 1980)

Anon, "User-Proofing Apple Programs," pg. 7-9.
Check your programs for common problems met by Apple users.

Smith, Bill, "ROM the Robot," pg. 12-19.
Part Two in a graphics program in Apple Lo-Res.

Nevin, Scott, "Lo-Res Printout," pg. 34-36.
Print out your Apple Lo-Res screen on your printer.

Truckenbrod, Joan, "Apple Graphics," pg. 85-88.
The second in a series of articles on Hi-Res graphics for the Apple. Notes on generation of moving figures.

826. L.A.U.G.H.S. 2, No. 3 (July, 1980 Supplement)

Connelly, Pat, "Shaper," pg. 3-7.
How to lift a shape or object off the Apple Hi-Res screen and place it in your shape tables.

827. Stems From Apple 3, Issue 7 (July, 1980)

Anon., "Day of Month," pg. 8-10.
A Pascal program for the Apple II.

International Apple Corps "APNOTES," pg. 15-19.
Several APNOTES including Notes on Random Number Generation, Print Using Simulator, HIRES SCREEN Function, Pascal Hi-Res Load/Save to Disk, etc.

828. Peek(65) 1, No. 7 (July, 1980)

Sanders, Jim, "A Visit to Monte Carlo," pg. 2.
A tutorial on random numbers with 2 examples for the OSI system.

Hawkins, Gary, "Conditional Control C."
Interrupt your listing with a graceful exit and nothing happening to your files.

Kourany, Paul, "Higher Resolution Graphics and Machine Language Save," pg. 7-9.
A routine for improving the resolution of OSI graphics and a machine language save routine.

Lew, Kevin G., "Morse Code Practice," pg. 10-12.
A code practice program for the OSI C2/4P using the audio out capabilities of the micro.

Atchley, Fred W., "The USR(X) Routine," pg. 19.
An example machine language program and BASIC listing to implement it.

829. Byte 5, No. 7 (July, 1980)

Hallgren, Richard C., "Interactive Control of a Videocassette Recorder with a Personal Computer," pg. 116-134.
Use of a computer to control videotaped material.

830. Peelings II 1, No. 2 (July/August, 1980)

Staff, "Apple Software Reviews," pg. 4-30.
22 new programs are reviewed in considerable detail.

831. Sym-Physis, Issue 4 (July/August, 1980)

Gieryic, Jack, "Notes on SYM-1 Programs," pg. 3-11.
Notes on several programs including Bill Gowan's Hi-Density Plot Routine, and a listing of a graphics program.

Gieryic, Jack, "Cursor Positioning/Graphics Primer," pg. 4-7.
A primer tutorial with two KIM-2 examples.

Gieryic, Jack, "Complex Sound Generator Chip," pg. 7-8.
The SN76477N, a $3 chip from Radio Shack, can be used for generating a lot of interesting sounds.

Winter, Frank, "TOPS - a Tape Operating System for the SYM," pg. 12-14.
All about TOPS, with a BASIC source to demonstrate a typical program to create a data file.

Hobart, Joe, "An EPROM Burner for the SYM-1," pg. 15-18.
Description of interface, hardware and software for TI 2516 and 2716, 2532, 2732 EPROMS.

832. Sym-Physis, Issue 4 (July/August, 1980)

Gschwind, Hans W., "Handling Data Files and Multi-Paramenter USR Functions," pg. 19-24.
Discussion of handling BASIC data files with listing and example of output.

833. Queue Catalogue, No. 2

Staff, "Educational Software for Apple II, PET and TRS-80," pg. 1-55.
A very complete listing of software (55 pages).

834. Compute 1, Issue 5 (July/August, 1980)

Staff, "Panasonic Microcomputer," pg. 5.
The new Panasonic is 6502-based with 1K, expansion module for 12K, very small and compact and about $400.

Budge, Joseph H., "Visicalc," pg. 19.
A major software review.

Lock, Robert, "Basically Useful BASIC," pg. 21.
Another article in this continuing series with examples of financial programs.

Deal, Elizabeth, "How to Program in BASIC with the Subroutine Power of FORTRAN," pg. 23-26.
A listing for the PET is given.

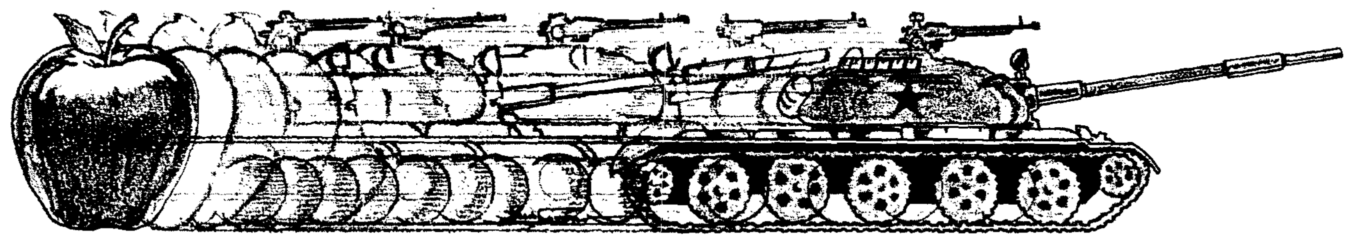
Baker, Al, "Programming Hints for Atari/Apple," pg. 34-36.
Joystick and Menu selection routines.

Schmoyer, Jeff, "Apple II ROM Card Documentation," pg. 49.
Circuit Diagram for the ROM Card and discussion thereof.

ADVERTISERS' INDEX

January 1981

Advertiser's Name	Page
Aardvark Technical Services	1, 78
Abacus Software	27
A B Computers	22
Adventure International	74
Alloy Engineering Company, Inc.	50
American Data, Inc.	20
Andromeda Inc.	15
Aurora Software Association	78
Avant-Garde Creations	50
Beta Computer Devices	26
CJM Industries	72
Classifieds	38
Computer Applications Tomorrow	16
The Computerist, Inc.	33, 40
Computers-R-Us	89
Creative Computing	54
Dakin5 Corporations	52
Decision Systems	52
Dr. Dobb's Journal	34
Dwo Quong Fok Lok Sow	81
Eastern House Software	50
E & I Technical Services	78
Enclosures Group	64
Excert, Inc.	47
Faragher Associates, Inc.	81
Galaxy	86
Hepburn, MCA	28
Image Computer	16, 16
Instant Software	62, 63
Lazer Systems	56
MICRO	28, 46, 47, 95
Mittendorf Engineering	78
Mountain Computer	IFC
Nibble	55
Nikrom Technical Products	86
Ohio Scientific	BC
OS Small Systems Journal	82-85
Omega Software Systems, Inc.	80
On-Line Systems	52
Optimized Data Systems	27
Peelings II	6
Pegasus Software	20
Perry Peripherals	28
Professional Software, Inc.	IBC
Programma International	4
Progressive Computer Software	46
Progressive Computing	81
Progressive Software	86
Rainbow Computing	86
Sirius Software	73, 74
Skyles Electric Works	27, 48, 49
Small Business Computer Systems	74
Soft CTRL Systems	19
Software Technology for Computers	73
Southeastern Software	2
Strategic Simulations	96
Sunset Electronics	20
Versa Computing, Inc.	60
Voicetek	27



HOW TO TURN AN APPLE INTO A TANK.

With **Computer Conflict**™ and a little imagination, we'll transform your staid and respectable Apple computer into the fearsome war machine of the Soviet Red Army. Computer Conflict actually consists of two fast-paced, action-packed war-games played on full-color mapboards of Hi-Res graphics: **Rebel Force** and **Red Attack!**

REBEL FORCE puts you in the role of a Soviet commander whose regiment must face a computer-directed guerrilla up-rising which has overrun a vital town. Armed with your tank, heavy-weapons, and infantry units, your mission is to regain the town through the annihilation of the Rebel Force.

Your advance will be brutally opposed by minefields, ambushes, militia, and anti-tank guns — all skillfully deployed by your computer. Survival and success of your units will depend on your ability to take advantage of the variable terrains - open, forest, and rough - each of which has different move-ment costs and shelter values.

In this finely-balanced solitaire wargame, every move is played under real-time conditions: Procrastinate and lose. At the same time, caution cannot be cast aside; severe unit losses will only result in a Pyhrric victory at best.

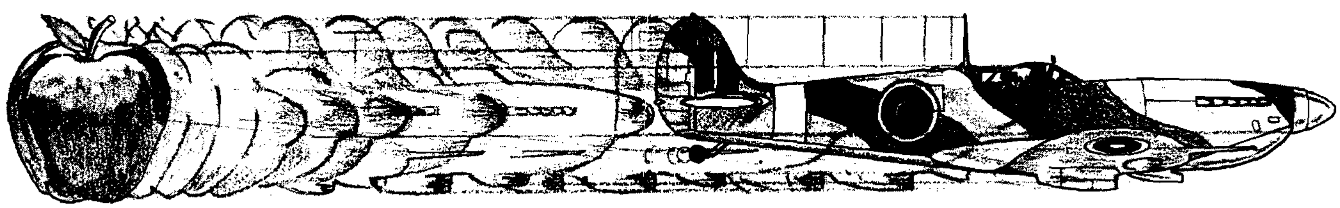
With its five levels of difficulty (plus one where you make up your own), the computer can and will stress your tactical skills to their fullest.

RED ATTACK! simulates an invasion by a mixed Soviet tank and infantry force against a defending battalion. As the defender, your task is to deploy your infantry units effectively to protect three crucial towns — towns that must not fall!

As the Russian aggressor, your objective is to crush the resistance by taking two of these three towns with your tanks and infantry. With control of these strongpoints, the enemy's capitulation is assured.

Red Attack! is a two-player computer simulation of modern warfare that adds a nice touch: At the start of each game, the computer displays a random setup of terrains and units, pro-viding every game with a new, challenging twist.

Computer Conflict, for $39.95, comes with the game pro-gram mini-disc and a rule book.



OR A SPITFIRE.

After you're done playing Computer Conflict, you may be in a mood for something other than ground-attack wargames. In that case, **Computer Air Combat**™ is just what you need.

With Computer Air Combat, your screen lights up with an open sky generated by Hi-Res graphics offering global and tactical plots. Squint your eyes a bit, let loose your mind, and you'd swear your keyboard has melted into the throttle, rudder, altimeter, and other cockpit instrumentation of a World War II combat plane. In fact, any of 36 famous fighters or bombers, from a Spitfire and B-17 Flying Fortress to the Focke-Wulf 190 and A6M5 Zero. Each plane is rated - in strict historical accuracy and detail - for firepower, speed, maneuverability, damage-tolerance, and climbing and diving ability.

Practically every factor involved in flying these magnificent airplanes has been taken into account, even down (or up?) to the blinding sun. Climb, dive, twist, and turn. Anything a real plane can do, you can do. However, the computer prevents all "illegal" moves - such as making an outside loop (which in real life, would disastrously stall a plane).

PLAY THE COMPUTER. Aside from being the game's perfect administrator and referee, the computer will serve as a fierce opponent in the solitaire scenarios provided: Dogfight, Bomber Formation, radar-controlled Nightfighter, and V-1 Inter-cept. There's even an Introductory Familiarization Flight (with Air Race option) to help you get off the ground.

With the number and type of planes and pilot ability variable, you can make the computer as challenging as you want to give you the ultimate flying experience.

PLAY A HUMAN. Two can play this game as well, in dogfights and bomber attacks. Given a handicap of more or better planes or an ace pilot (or all of the above), even a novice at Computer Air Combat stands a chance to defeat a battle-hardened veteran.

For $59.95, Computer Air Combat gives you the game disc, a rule book, two mapboard charts (for plotting strategies between moves), and three player-aid charts.

Credit card holders, if you own an Apple®II 48K (Apple-soft ROM) and a mini-floppy disc drive, call **800-227-1617 ext. 335 (toll free)** and charge your order to your VISA or MASTERCHARGE. In California, call 800-772-3545, ext. 335.

To order by mail, send your check to:
Strategic Simulations Inc, Dept. PC, 465 Fairchild Drive, No. 108, Mountain View, CA 94043. All our games carry a 14-day money back guarantee to assure your satisfaction.

While you're at it, you can also get our other games:

- ☐ **Computer Bismarck** for your Apple: $59.95
  Computer Bismarck, TRS-80® 48K Disc: $59.95
  Computer Bismarck, TRS-80 32K Cassette: $49.95
- ☐ **Computer Ambush** (a tactical simulation of man-to-man combat in WWII) for your apple: $59.95
- ☐ **Computer Napoleonics**, the Battle of Waterloo for your Apple: $59.95
- ☐ **Computer Quarterback** (a real-time strategy football game): $39.95

IF YOU'RE WAITING FOR THE PRICE OF WORD PROCESSORS TO FALL WITHIN REASON,

Everyone expected it would happen sooner or later...with WordPro PLUS™ it already has! Now all the marvelous benefits of expensive and advanced word processing systems are available on Commodore computers, America's largest selling computer line. WordPro PLUS, when combined with the new 80 column CBM 8032, creates a word processing system comparable to virtually any other top quality word processor available—but at savings of thousands of dollars!

New, low cost computer technology is now available at a fraction of what you would expect to pay. This technology allowed Commodore to introduce the new and revolutionary CBM 8032 Computer.

WordPro PLUS turns this new CBM 8032 Computer into a sophisticated, time saving word processing tool. With WordPro PLUS, documents are displayed on the computer's screen. Editing and last minute revisions are simple and easy. No more lengthy re-typing sessions. Letters and documents are easily re-called from memory storage for editing or printing with final drafts printed perfectly at over five hundred words per minute!

Our nationwide team of professional dealers will show you how your office will benefit by using WordPro PLUS. At a price far less than you realize.

Invest in your office's future...
Invest in WordPro PLUS...
Call us today for the name of the WordPro PLUS dealer nearest you.

Professional Software Inc.
166 Crescent Road
Needham, MA 02194
(617) 444-5224
TELEX: 95 1579

The home computer you thought was years away is here.

C8P DF

Ohio Scientific's top of the line personal computer, the C8P DF. This system incorporates the most advanced technology now available in standard configurations and add-on options. The C8P DF has full capabilities as a personal computer, a small business computer, a home monitoring security system and an advanced process controller.

Personal Computer Features
The C8P DF features ultra-fast program execution. The standard model is twice as fast as other personal computers such as the Apple II and PET. The computer system is available with a GT option which nearly doubles the speed again, making it comparable to high end mini-computer systems. High speed execution makes elaborate video animation possible as well as other I/O functions which until now, have not been possible. The C8P DF features Ohio Scientific's 32 x 64 character display with graphics and gaming elements for an effective resolution of 256 x 512 points and up to 16 colors. Other features for personal use include a programmable tone generator from 200 to 20KHz and an 8 bit companding digital to analog converter for music and voice output, 2-8 axis joystick interfaces, and 2-10 key pad interfaces. Hundreds of personal applications, games and educational software packages are currently available for use with the C8P DF.

Business Applications
The C8P DF utilizes full size 8" floppy disks and is compatible with Ohio Scientific's advanced small business operating system, OS-65U and two types of information management systems, OS-MDMS and OS-DMS. The computer system comes standard with a high-speed printer interface and a modem interface. It features a full 53-key ASCII keyboard as well as 2048 character display with upper and lower case for business and word processing applications.

Home Control
The C8P DF has the most advanced home monitoring and control capabilities ever offered in a computer system. It incorporates a real time clock and a unique FOREGROUND/BACKGROUND operating system which allows the computer to function with normal BASIC programs at the same time it is monitoring external devices. The C8P DF comes standard with an AC remote control interface which allows it to control a wide range of AC appliances and lights remotely without wiring and an interface for home security systems which monitors fire, intrusion, car theft, water levels and freezer temperature, all without messy wiring. In addition, the C8P DF can accept Ohio Scientific's Votrax voice I/O board and/or Ohio Scientific's new universal telephone interface (UTI). The telephone interface connects the computer to any touch-tone or rotary dial telephone line. The computer system is able to answer calls, initiate calls and communicate via touch-tone signals, voice output or 300 baud modem signals. It can accept and decode touch-tone signals, 300 baud modem signals and record incoming voice messages. These features collectively give the C8P DF capabilities to monitor and control home functions with almost human-like capabilities.

Process Controller
The C8P DF incorporates a real time clock, FOREGROUND/BACKGROUND operation and 16 parallel I/O lines. Additionally a universal accessory BUS connector is accessible at the back of the computer to plug in additional 48 lines of parallel I/O and/or a complete analog signal I/O board with A/D and D/A and multiplexers.

Clearly, the C8P DF beats all existing small computers in conventional specifications plus it has capabilities far beyond any other computer system on the market today.

C8P DF is an 8-slot mainframe class computer with 32K static RAM, dual 8" floppies, and several open slots for expansion.

Prices start at under $3,000.

Computers come with keyboards and floppies where specified. Other equipment shown is optional.

For literature and the name of your local dealer, CALL 1-800-321-6850 TOLL FREE.

OHIO SCIENTIFIC
1333 SOUTH CHILLICOTHE ROAD
AURORA, OH 44202 • (216) 831-5600